

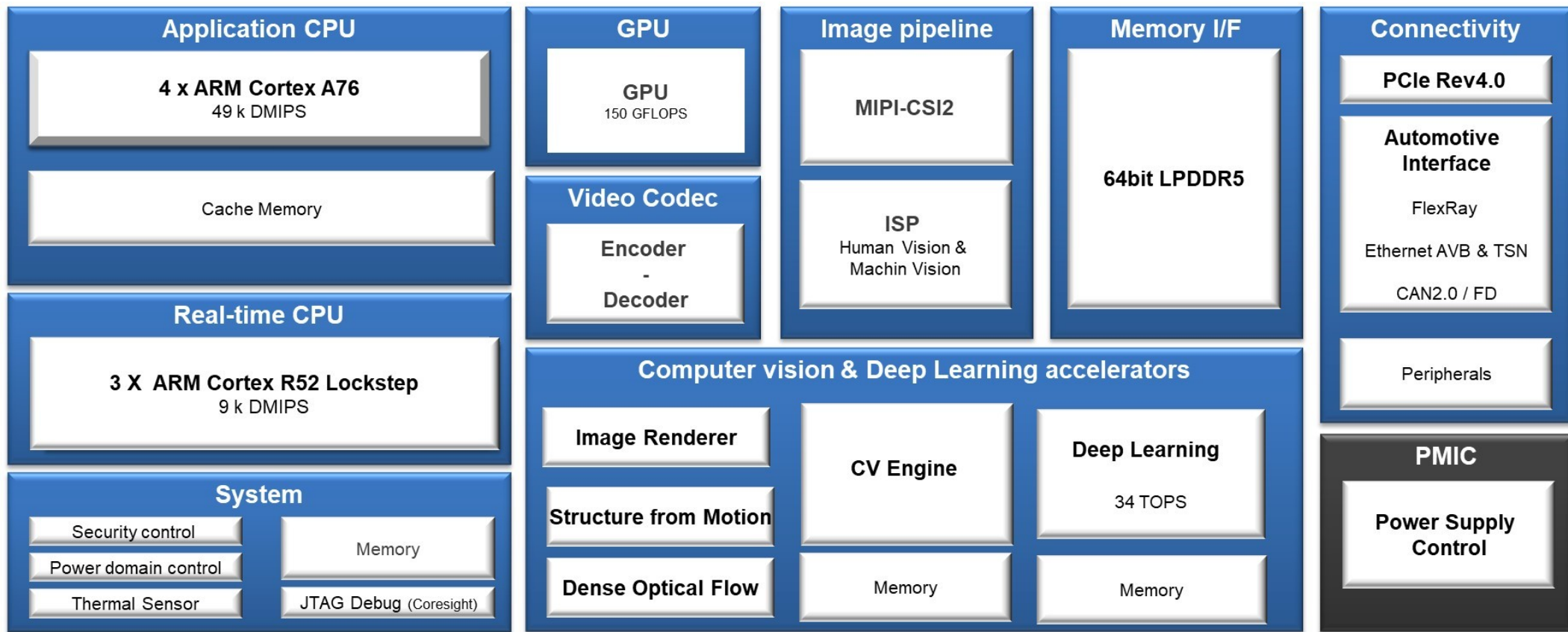


# Learn How to Support Your SoC and ISP in libcamera

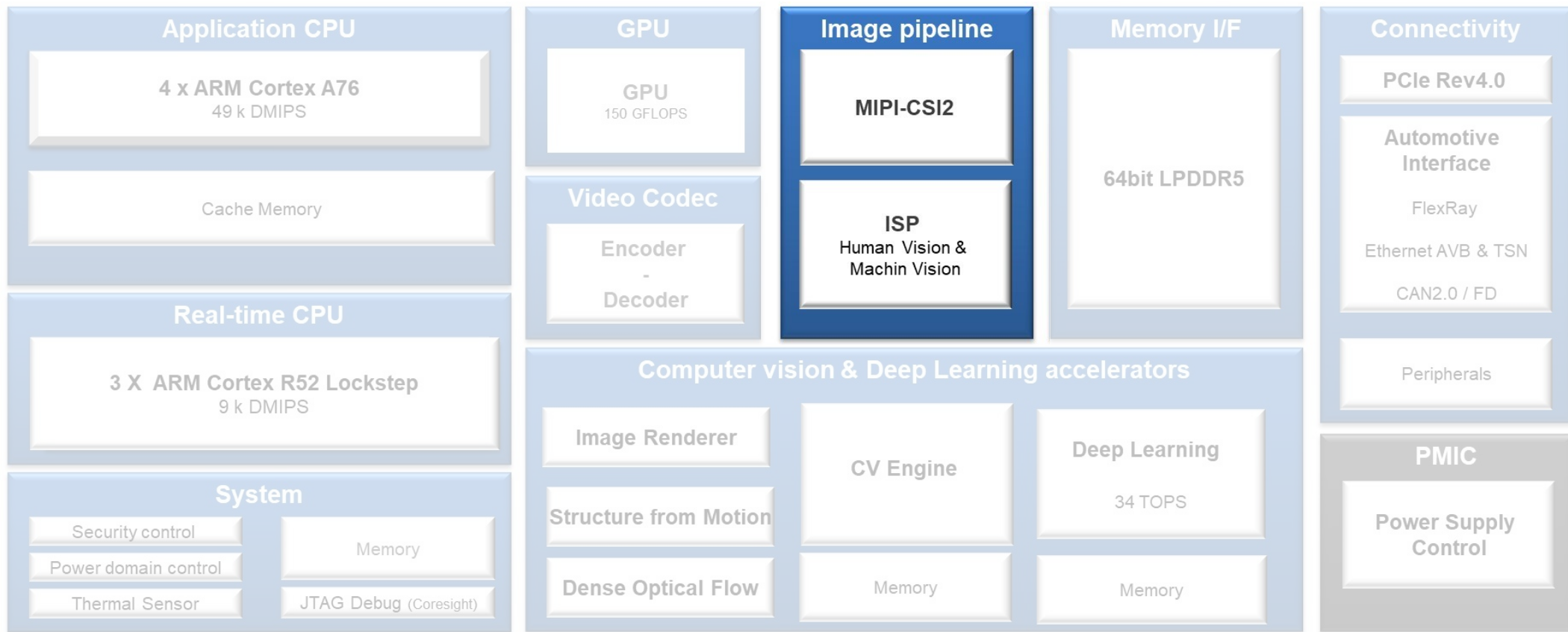
Embedded Linux Conference 2023  
Prague, 2023-06-30

Laurent Pinchart  
[laurent.pinchart@ideasonboard.com](mailto:laurent.pinchart@ideasonboard.com)

IDEAS  
ON BOARD



# An Amazing SoC



# With The Best Camera Hardware









# The Broccoli Pi





# Bring Your Camera to 2018: Forward Porting Image Sensor Drivers

Jacopo Mondì, Renesas



0:00 / 32:30 • Introduction >

Bring Your Camera into 2018: Forward Porting Image Sensor Drivers - Jacopo Mondì



The Linux Foundation  
16.6K subscribers

Subscribe

60



Share

Save



<https://www.youtube.com/watch?v=PJVlvUf0gP4>





# Image Signal Processing (ISP) Drivers & How to Merge One Upstream

Helen Koike / Collabora

#ossummit @HelenFornazier



0:00 / 52:57



Image Signal Processing (ISP) Drivers & How to Merge One Upstream - Helen Koike, Collabora



The Linux Foundation  
168K subscribers

Subscribe

37



Share

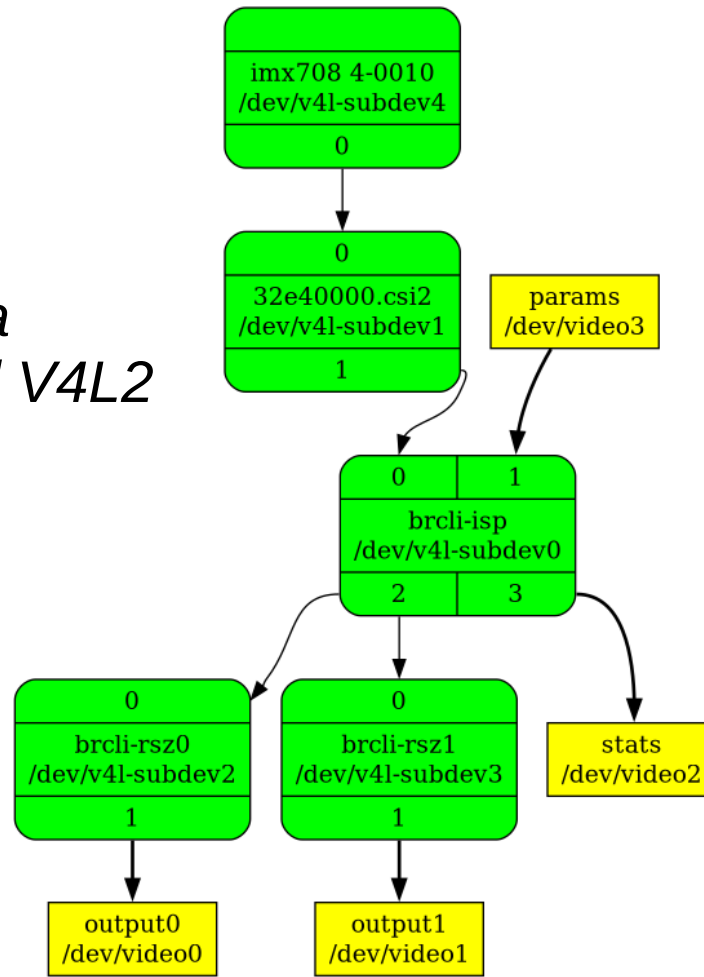
Save



<https://www.youtube.com/watch?v=SuAiJOtCxQY>



*Use the Media  
Controller and V4L2  
APIs*

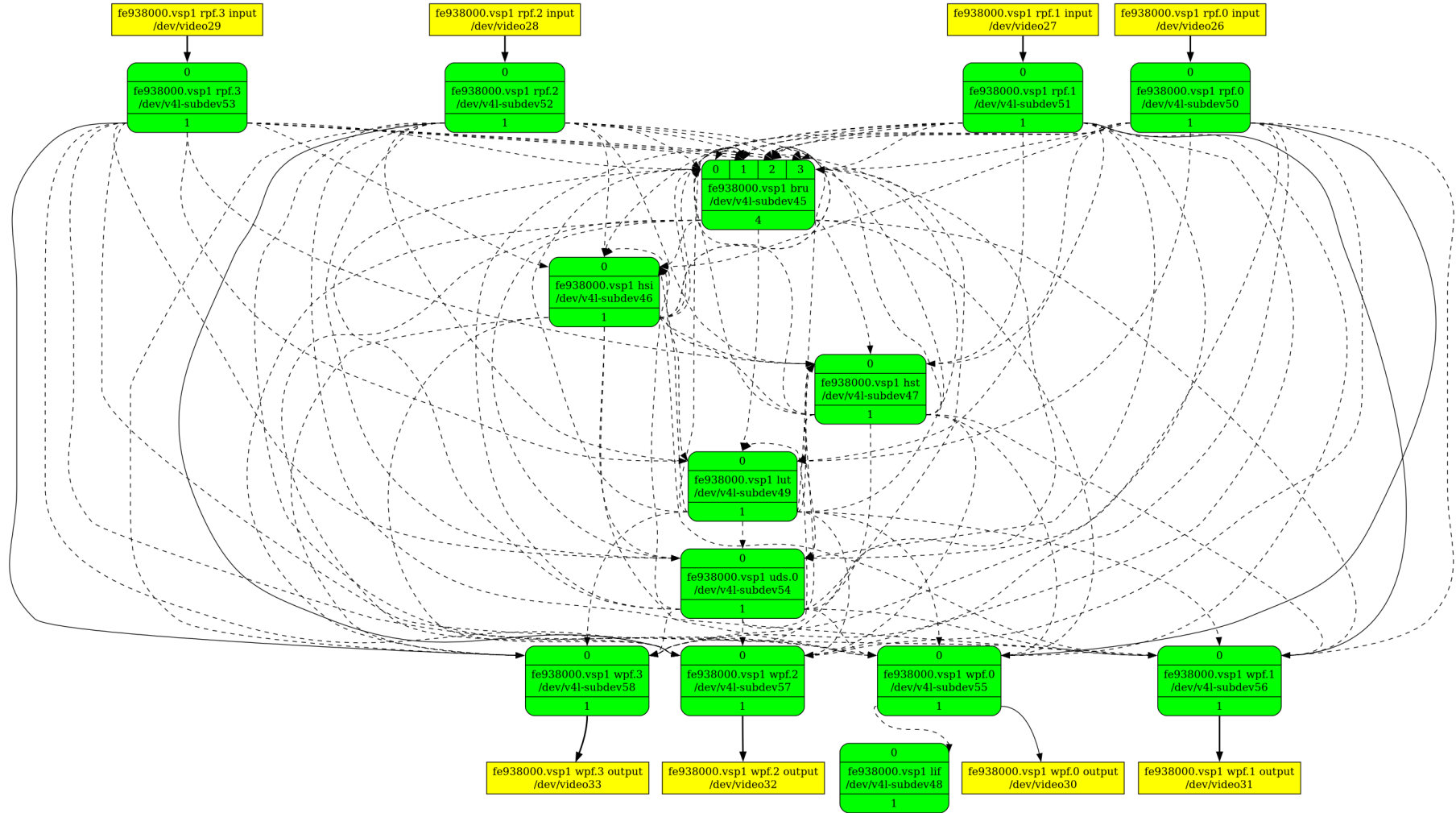


*Document and  
upstream the drivers*

*The community can  
provide support*

# Kernel Drivers





# Complex Hardware



# A Bitter Tasting Broccoli



# Why Embedded Cameras are Difficult, and How to Make Them Easy

Laurent Pinchart / Ideas on Board



0:00 / 46:21



Why Embedded Cameras are Difficult, and How to Make Them Easy - Laurent Pinchart, Ideas on Board



The Linux Foundation  
168K subscribers

Subscribe

79



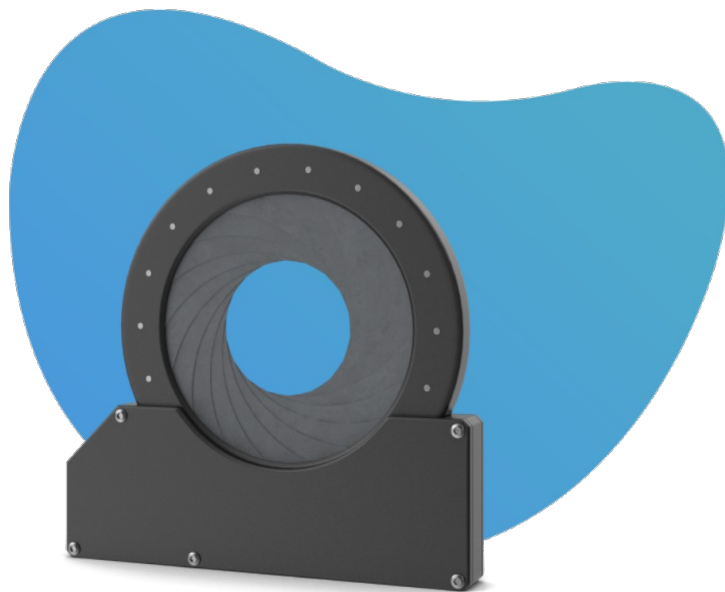
Share

Save



<https://www.youtube.com/watch?v=GIhV7tiUji0>





# Hi, we're **libcamera.**

An open source camera stack and framework for Linux, Android, and ChromeOS

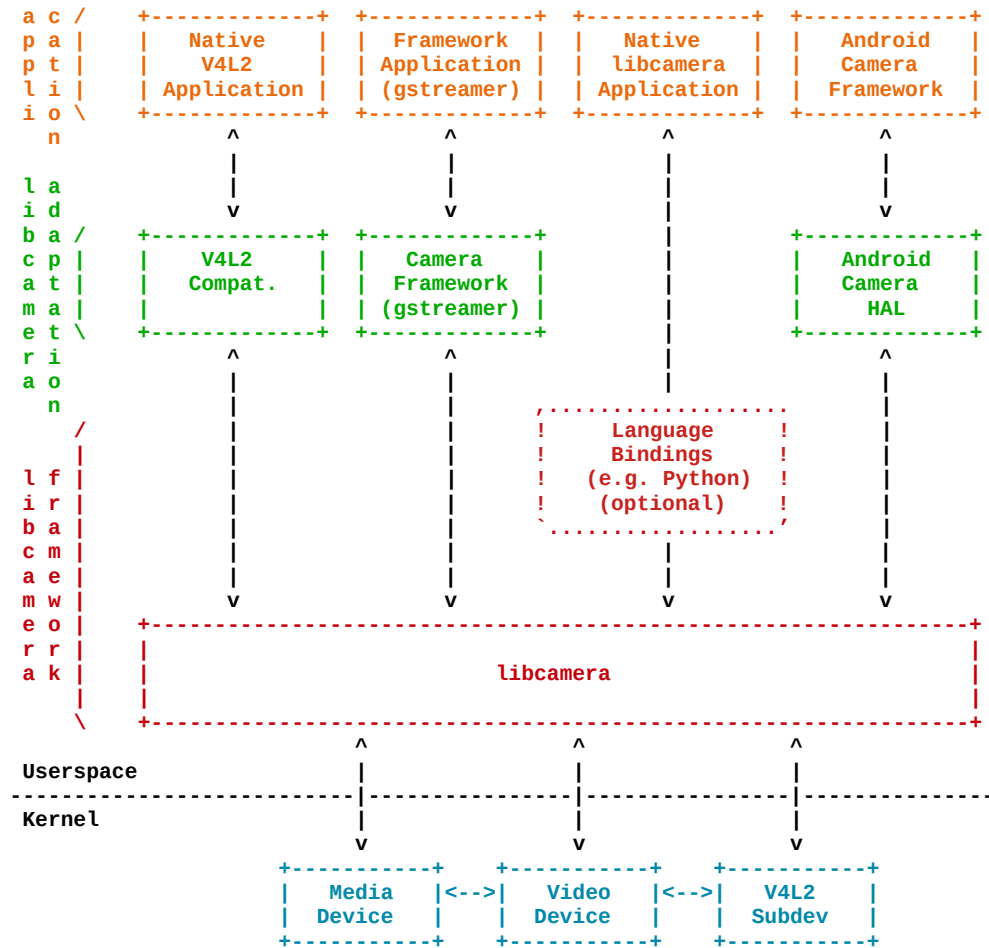
[Getting Started](#)

**IDEAS**  
ON BOARD

+ - / \ - +  
| (o) |  
+ - - - - +

libcamera

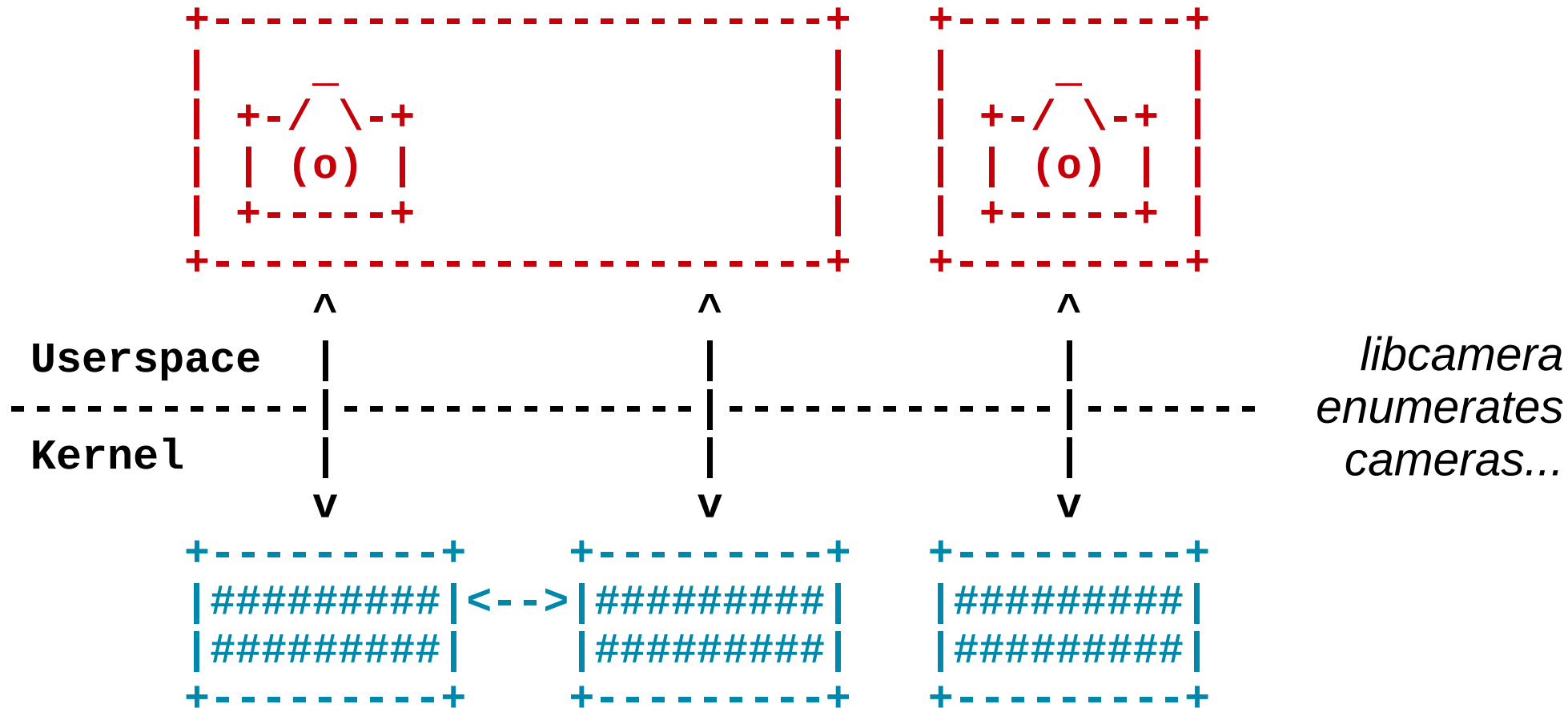
*libcamera provides a complete userspace camera stack.*



*The 'Mesa' of the camera world.*

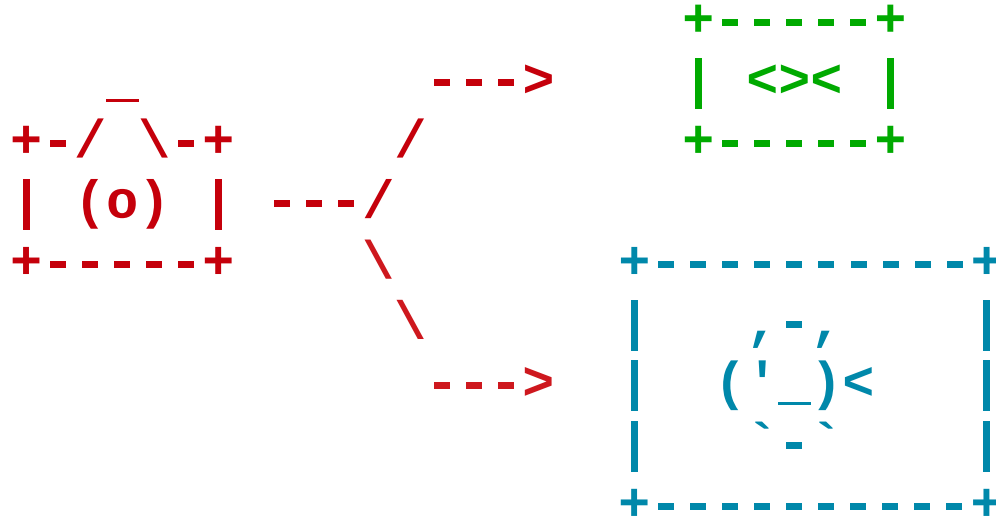
# Camera Stack





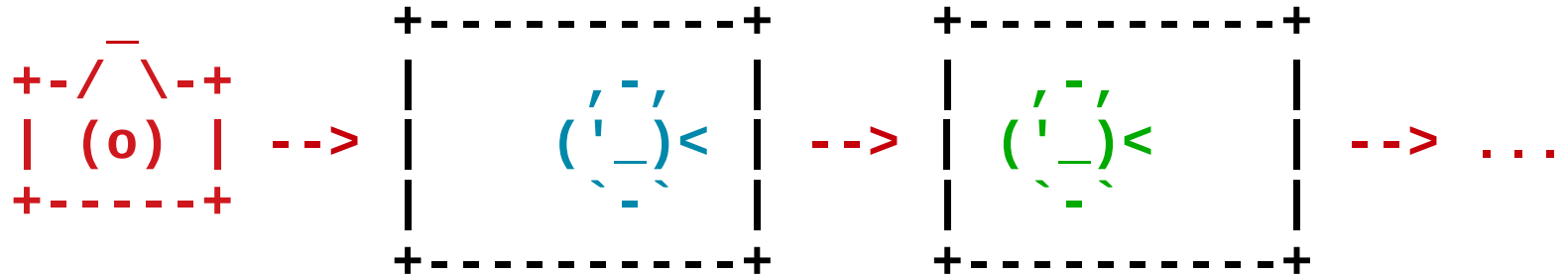
# Camera Devices & Enumeration





*It supports multiple  
concurrent streams  
for the same  
camera...*

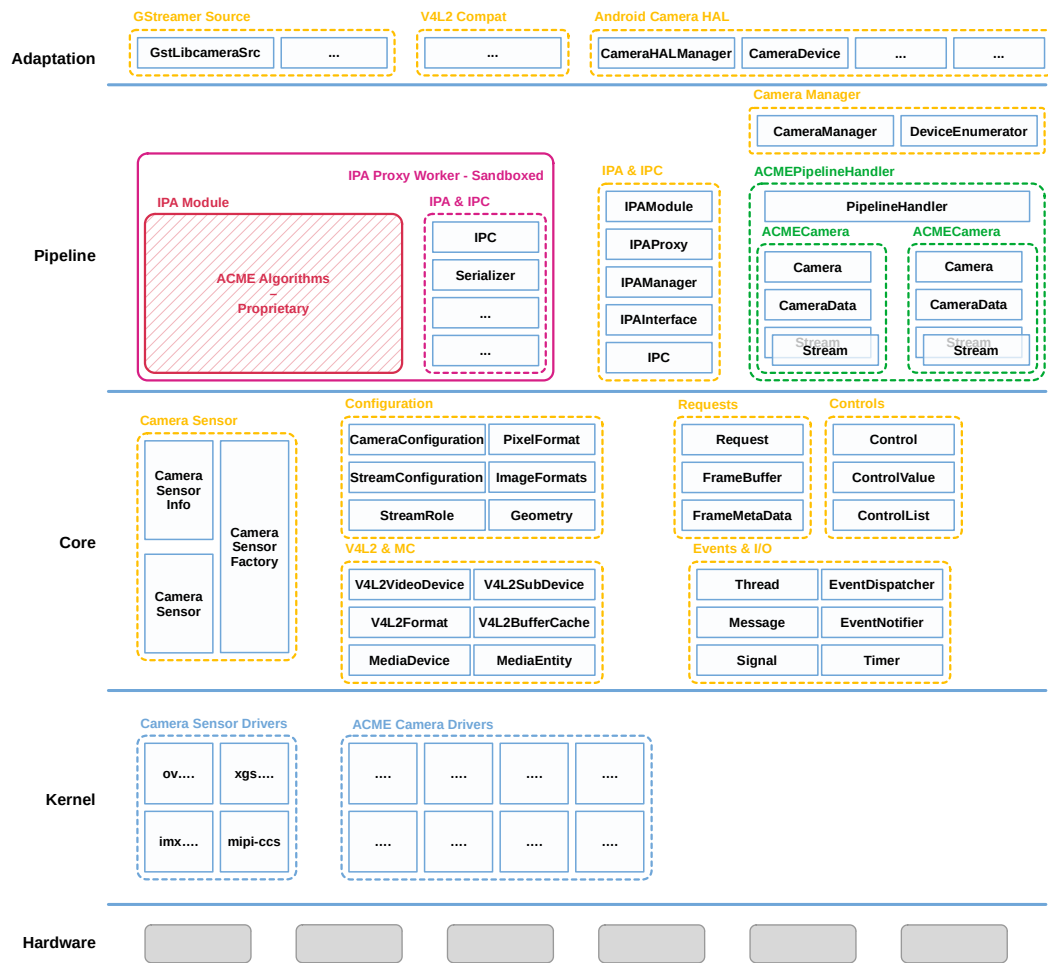
# Streams



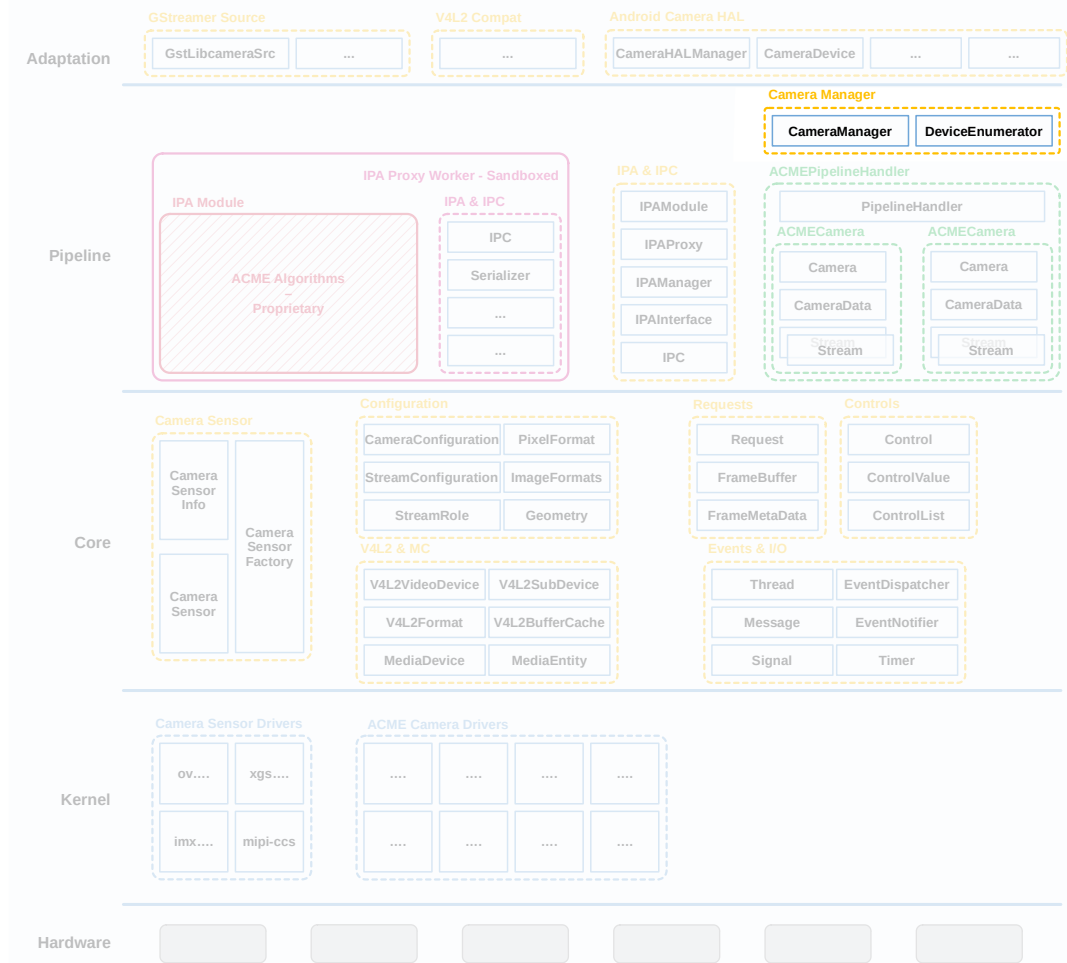
*... and per-frame controls.*

## Per-Frame Controls



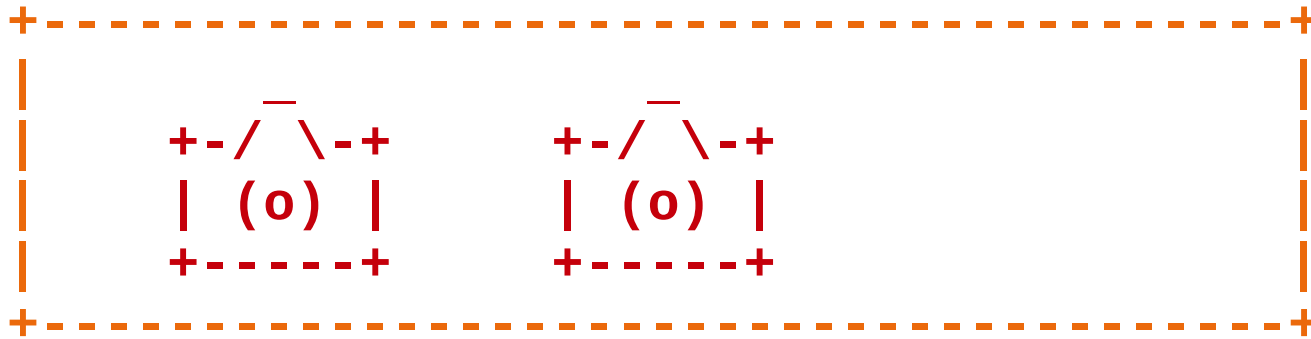


# The Camera Stack



# The Camera Manager

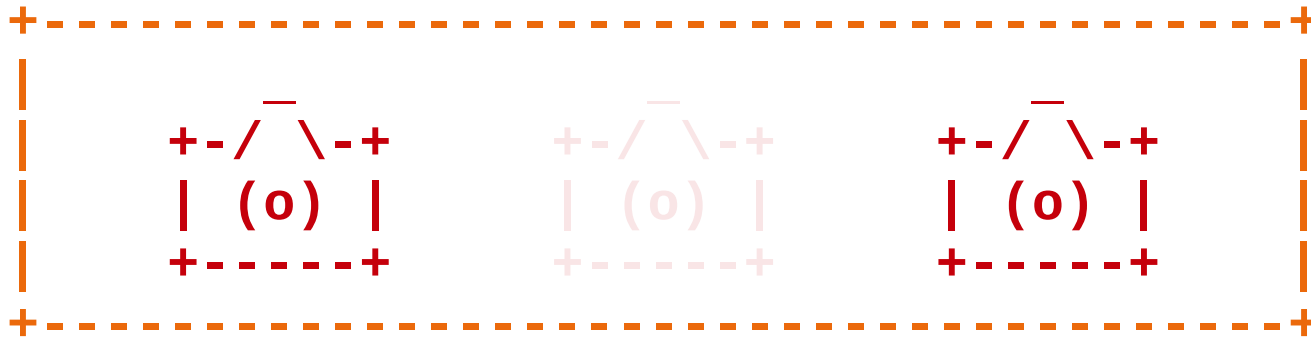




*The Camera Manager  
enumerates media  
devices and instantiates  
corresponding pipeline  
handlers.*

# The Camera Manager

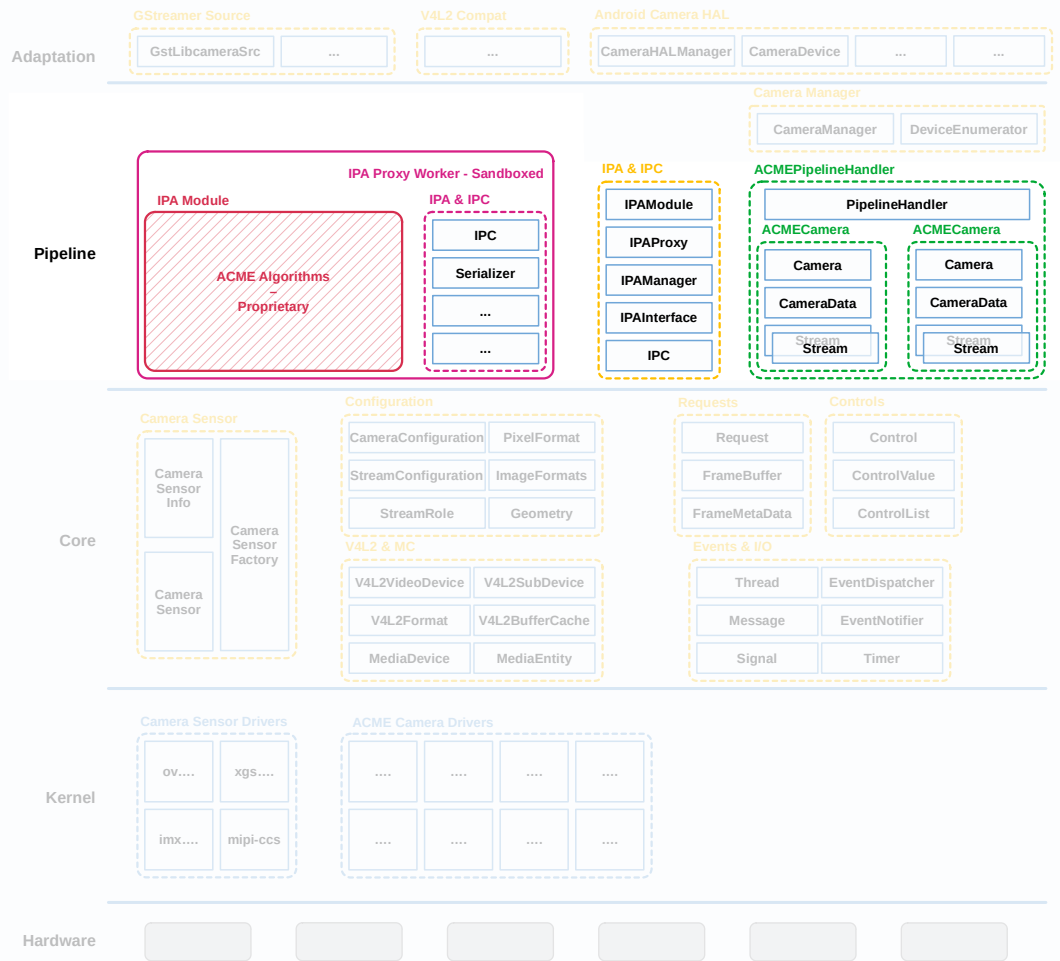




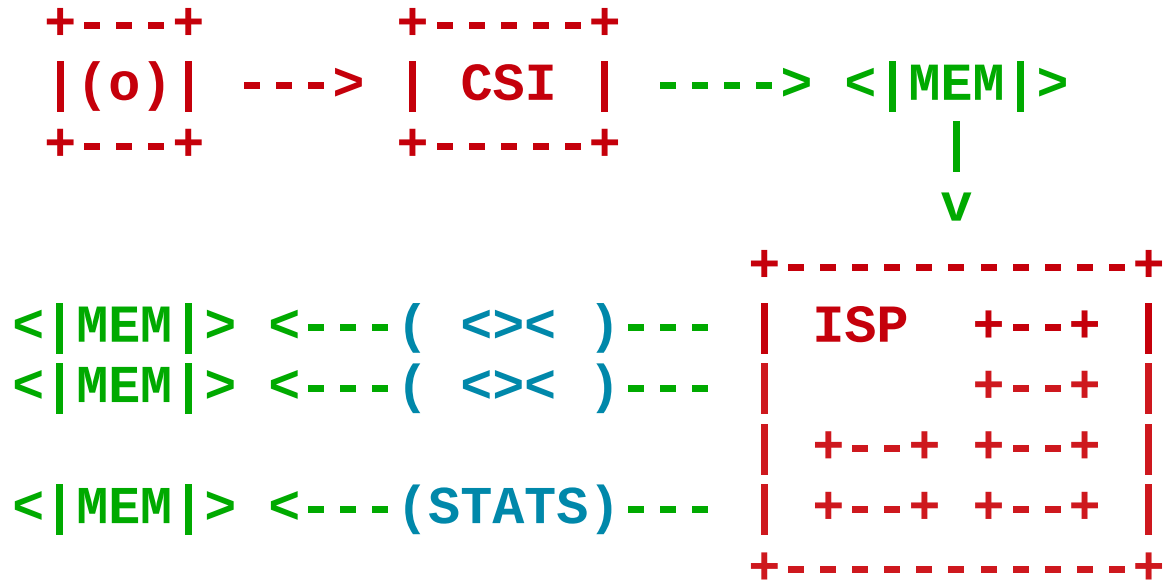
*Each pipeline handlers create and register one or more cameras.*



# The Camera Manager



# The Pipeline

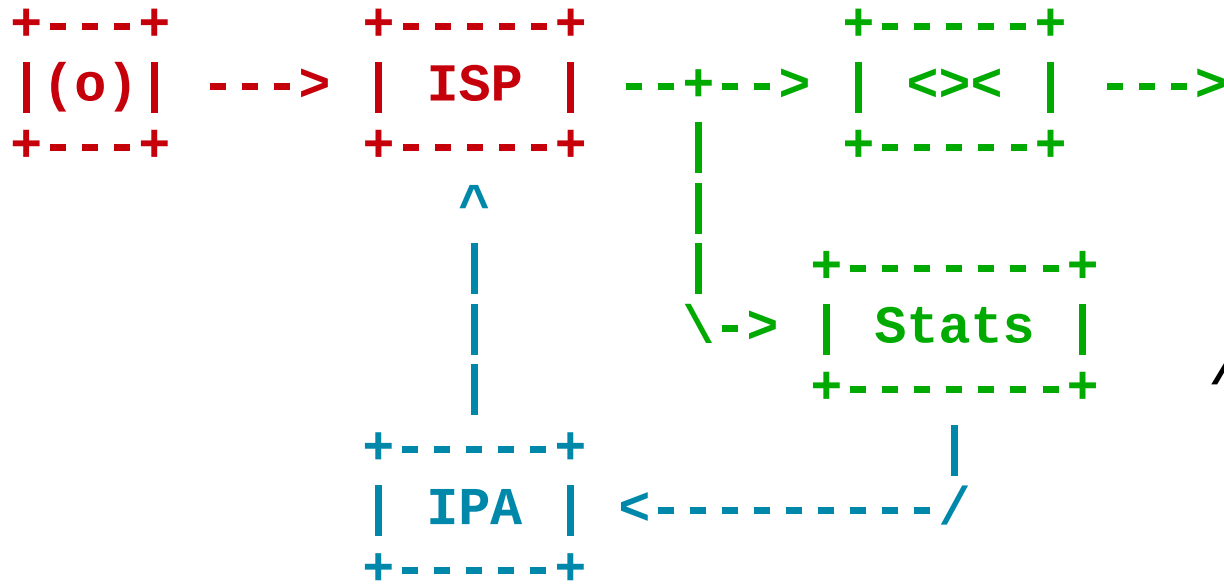


*The pipeline handler interfaces with all kernel devices. It abstracts them and exposes video streams to upper layers.*



# The Pipeline Handler

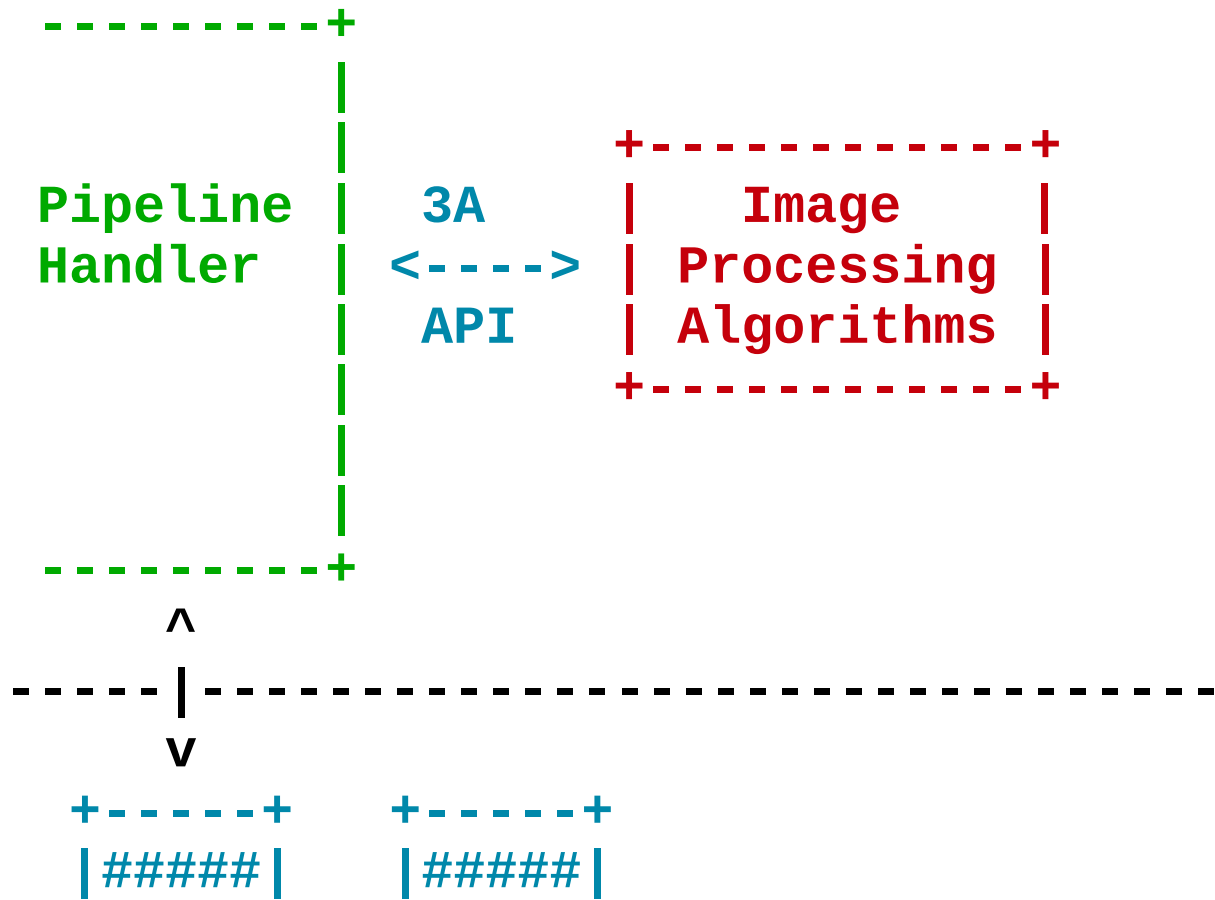




*Image Processing Algorithms (IPA) receive statistics from the hardware and compute optimal image parameters.*

# The Image Processing Algorithms

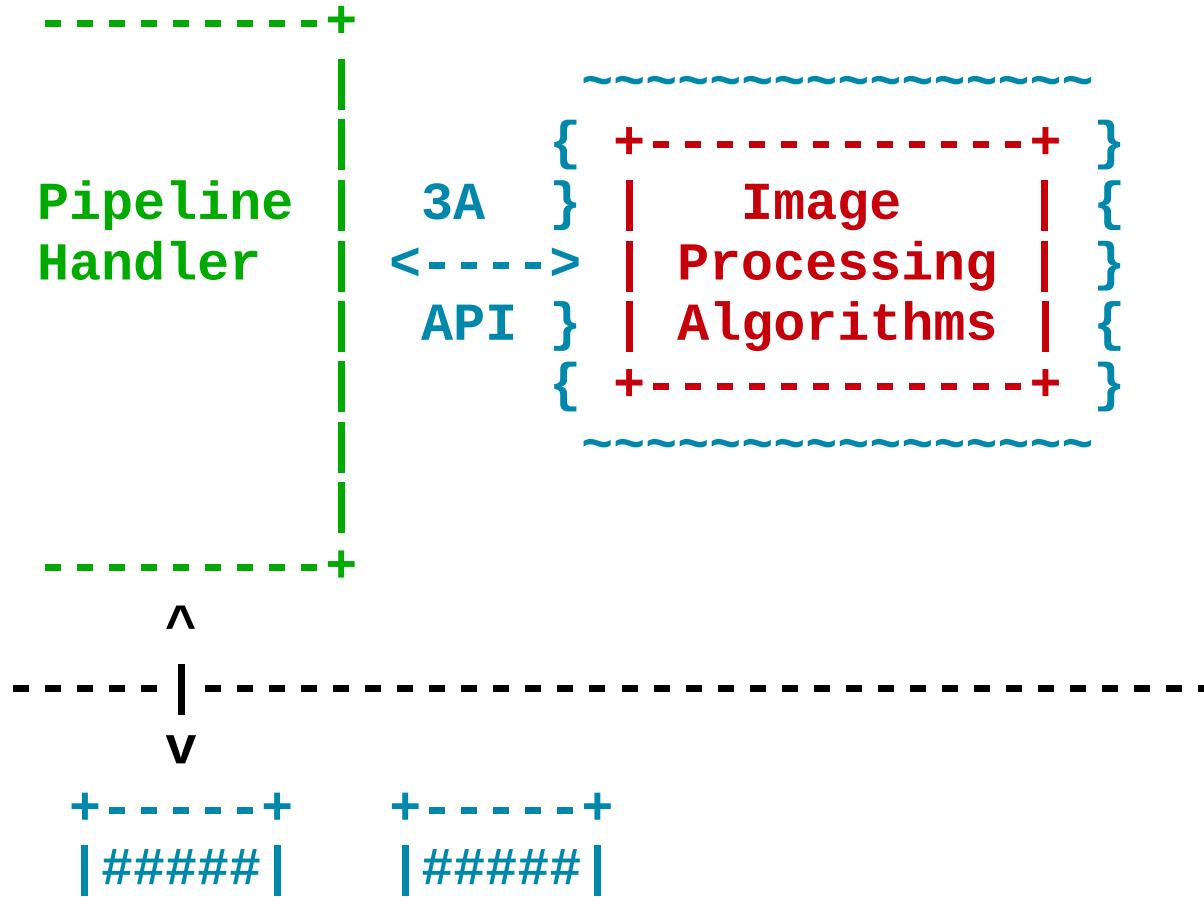




*IPAs are separate modules that don't access kernel devices directly. They only have access to their pipeline handler through the IPA API.*

# The Image Processing Algorithms

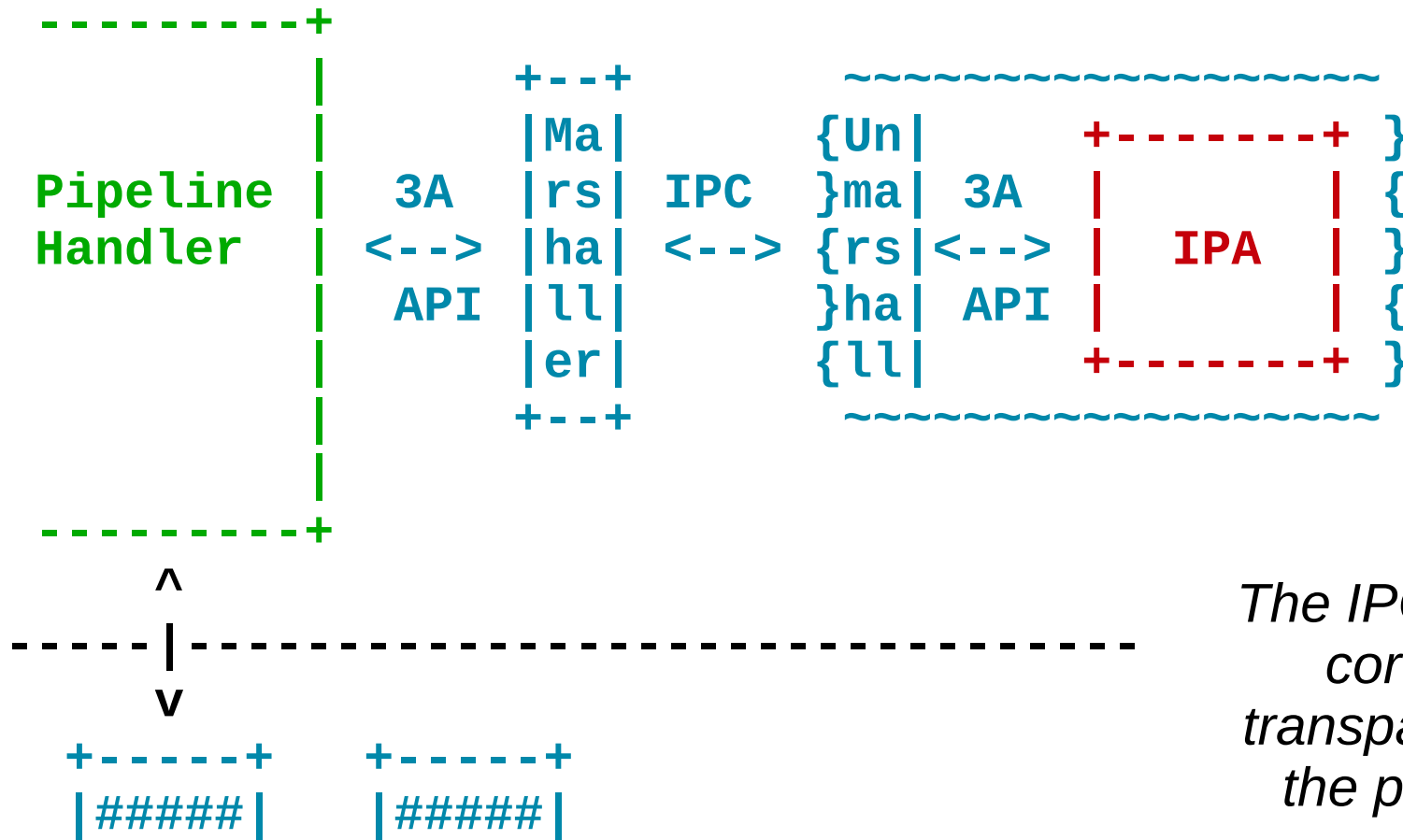




*Out-of-tree (including closed-source) IPAs are sandboxed in a separate process. They communicate with the pipeline handler through IPC.*

# The Image Processing Algorithms





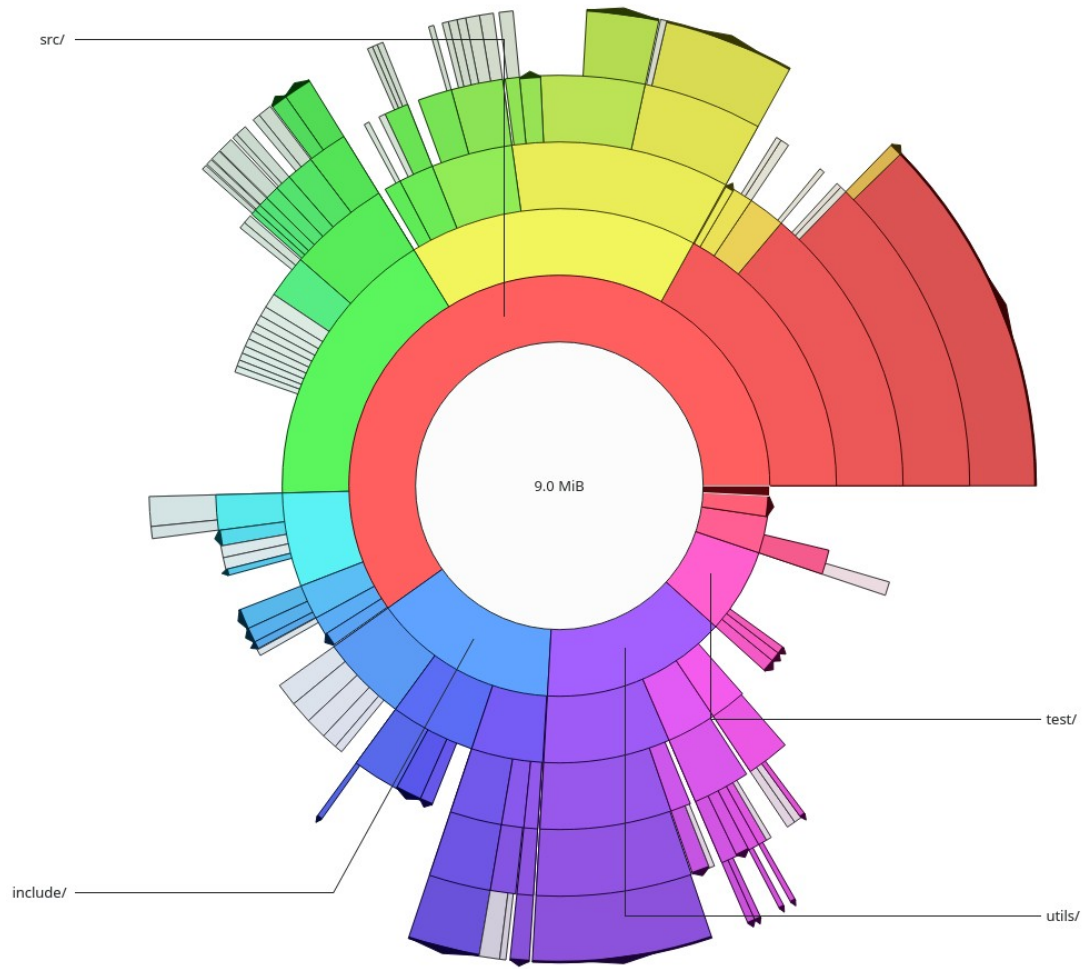
*The IPC is handled in core components, transparently for both the pipeline handler and the IPA.*

# The Image Processing Algorithms

+ - / \ - +  
| (o) |  
+ - - - - +

libcamera  
source tree  
(for reference only)





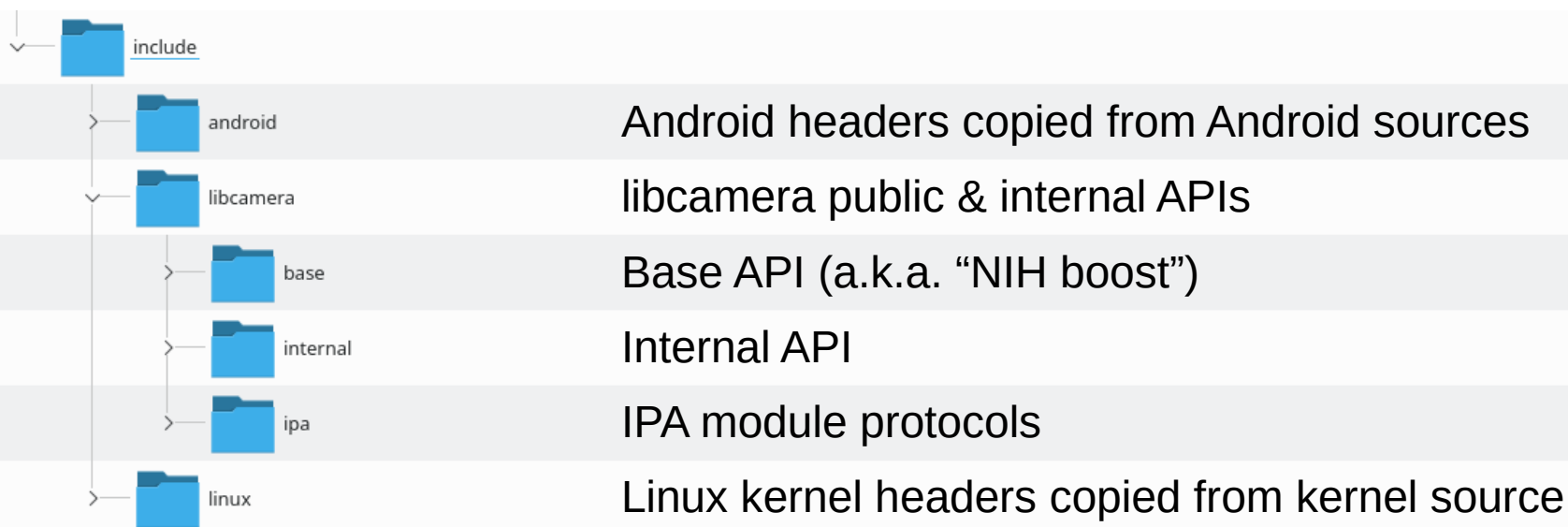
# Navigating in libcamera

> Documentation	Documentation, guides, tutorials, ...
> include	Headers (public API, internal API, ...)
> LICENSES	Licenses
> package	Distribution packaging
> <u>src</u>	Source
> subprojects	Meson subprojects
> test	Unit tests
> utils	Miscellaneous utilities



# Project Structure



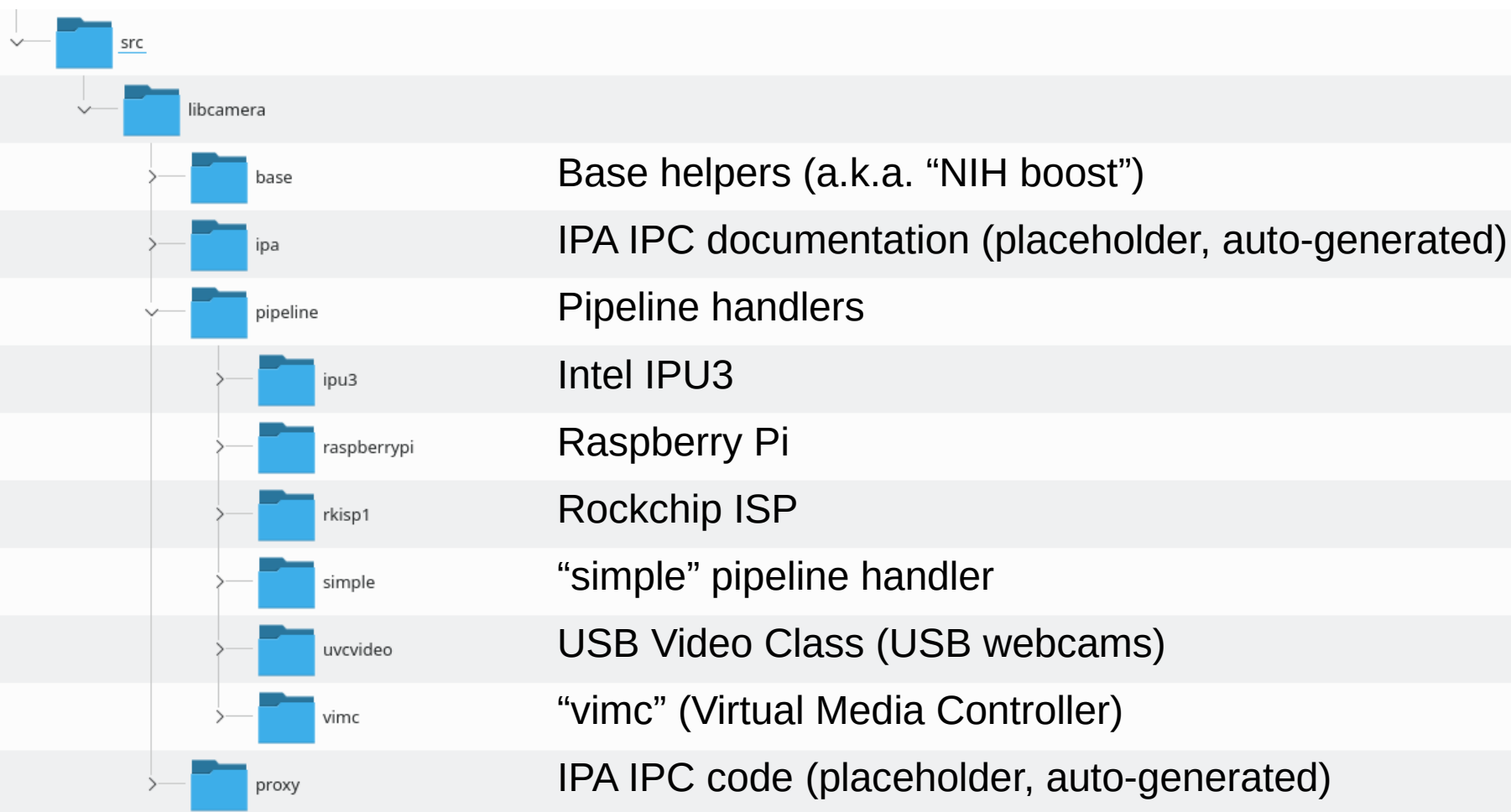


## Project Structure – /include

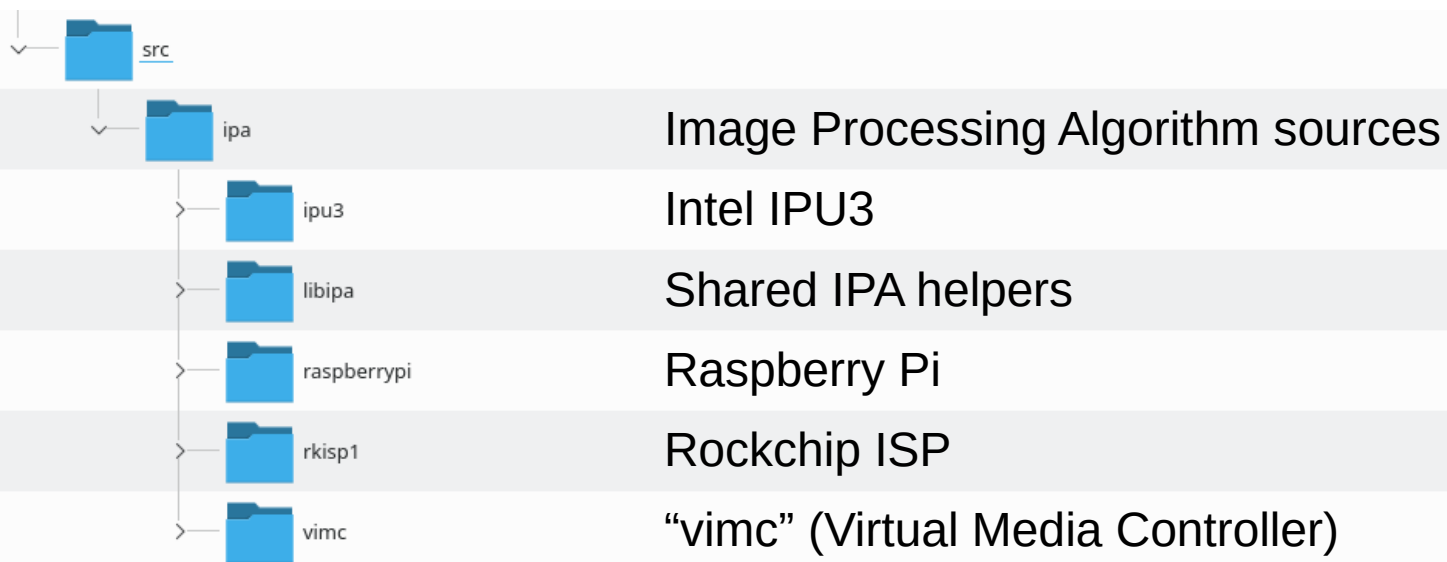
src	
android	Android camera HAL
apps	test applications (CLI, GUI, compliance test)
gststreamer	GStreamer source element
ipa	Image Processing Algorithm modules
libcamera	libcamera core
py	Python bindings and sample applications
v4l2	v4l2 compatibility



## Project Structure – /src



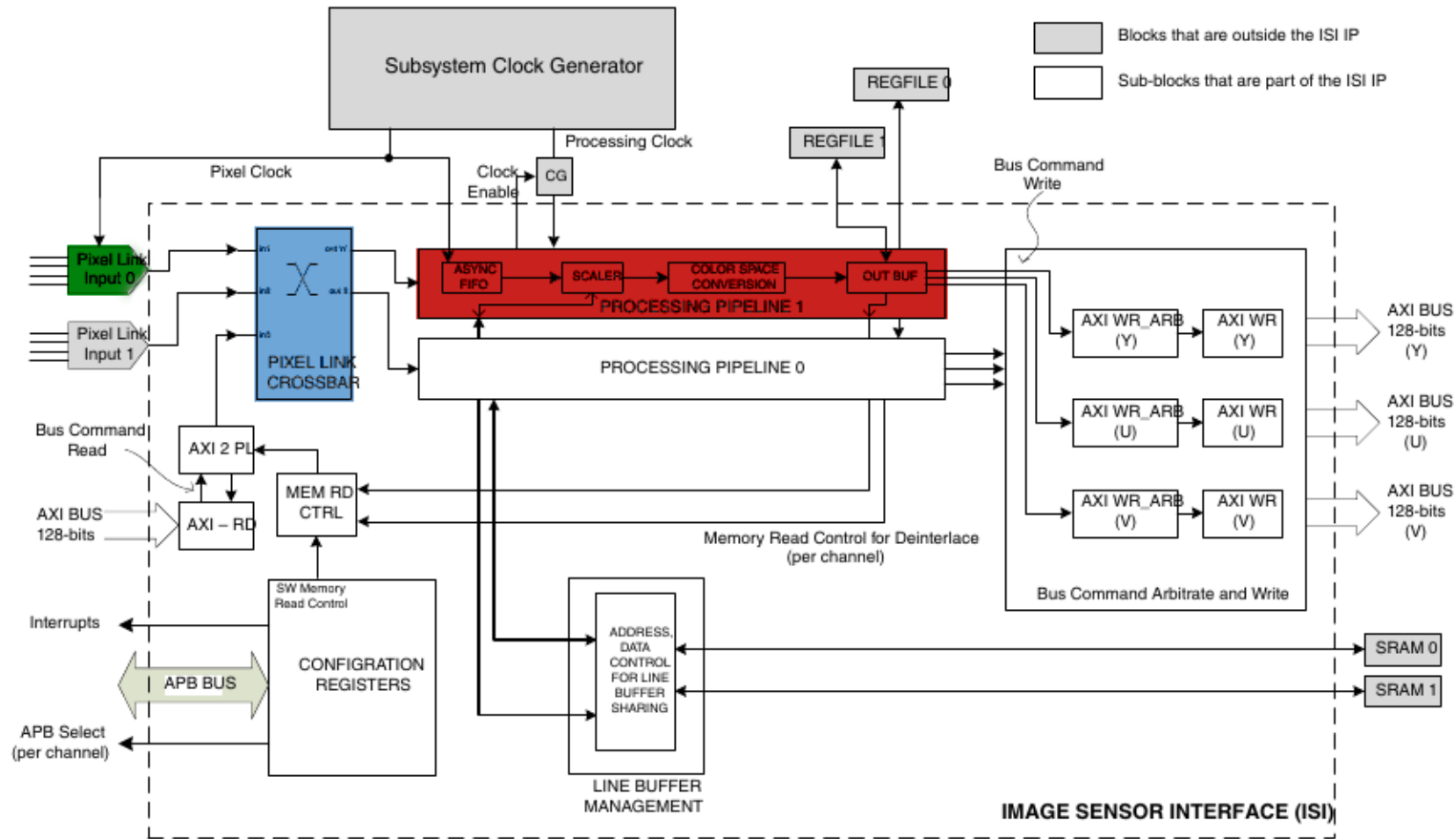
## Project Structure – /src/libcamera



## Project Structure – `/src/ipa`

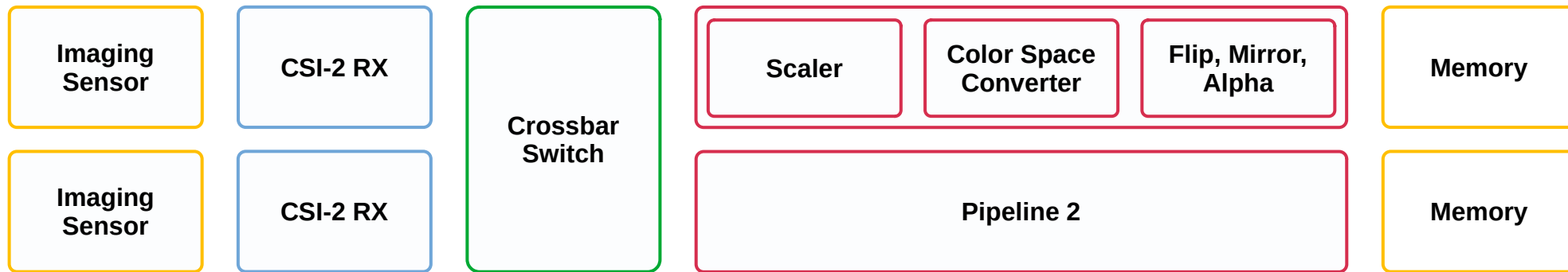
+ - / \ - +  
| (o) |  
+ - - - - +

Let's start  
simple(r)



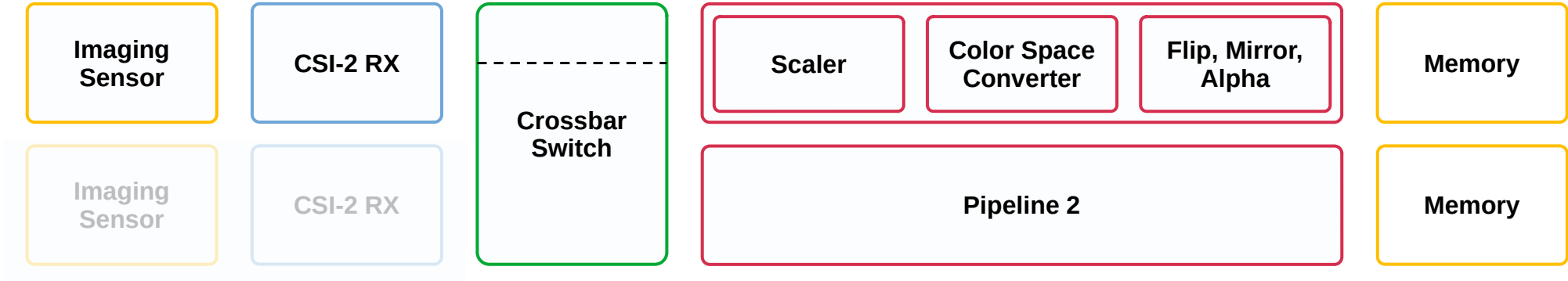
i.MX 8M Plus Applications Processor Reference Manual, Rev. 1, 06/2021

# Image Sensing Interface (ISI)



# i.MX8MP Image Sensing Interface (ISI)





# First Use Case: ISI, Single Camera

+ - / \ - +  
| (o) |  
+ - - - - +

The simple  
pipeline  
handler





The “simple” pipeline handler supports simple pipelines through fully generic code. It is probably the most complicated pipeline handler implementation in libcamera.

A simple pipeline is defined as a linear pipeline without any processing block that requires specific configuration.

← here

Additionally, a generic V4L2 M2M device may be used to perform scaling and format conversion.

# The Simple Pipeline Handler



## \* Overview

\* -----

\*

\* The SimplePipelineHandler relies on generic kernel APIs to control a camera device, without any device-specific code and with limited device-specific static data.

\*

\* To qualify for support by the simple pipeline handler, a device shall

\*

- \* - be supported by **V4L2 drivers**, exposing the **Media Controller API**, the **V4L2 subdev APIs** and the media bus format-based enumeration extension for the VIDIOC\_ENUM\_FMT ioctl ;
- \* - **not** expose any **device-specific API** from drivers to userspace ;
- \* - include **one or more camera sensor** media entities and **one or more video capture devices** ;
- \* - have a **capture pipeline with linear paths** from the camera sensors to the video capture devices ; and
- \* - have an optional memory-to-memory device to perform format conversion and/or scaling, exposed as a V4L2 M2M device.

src/libcamera/pipeline/simple/simple.cpp



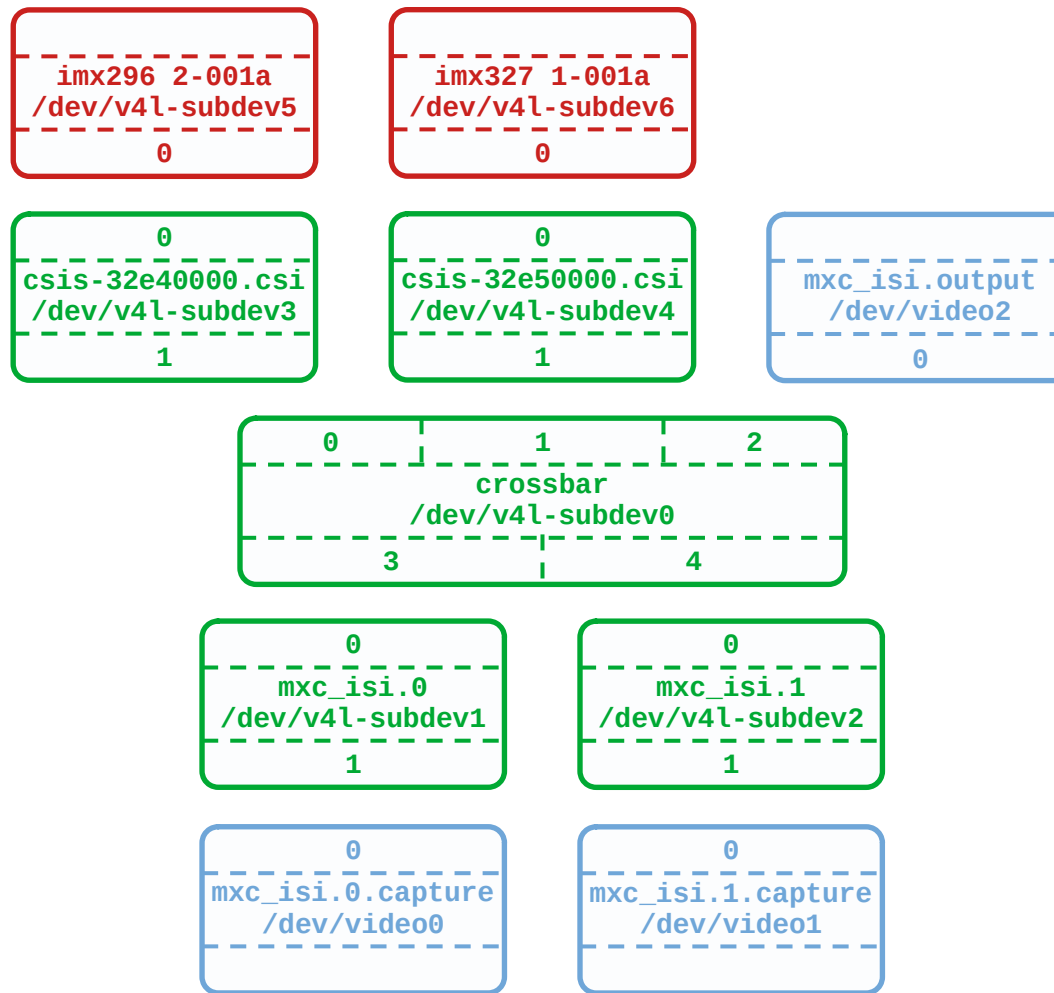
# The Simple Pipeline Handler

\* As devices that require a specific pipeline handler may still match the  
\* above characteristics, the simple pipeline handler doesn't attempt to  
\* automatically determine which devices it can support. It instead relies on  
\* an **explicit list of supported devices**, provided in the supportedDevices  
\* array.  
\*  
\* When matching a device, the pipeline handler **enumerates all camera sensors**  
\* and attempts, for each of them, to **find a path to a video capture video node**.  
\* It does so by using a **breadth-first search** to find the shortest path from the  
\* sensor device to a valid capture device. This is guaranteed to produce a  
\* valid path on devices with one only option and is a good heuristic on more  
\* complex devices to skip paths that aren't suitable for the simple pipeline  
\* handler. For instance, on the IPU-based i.MX6, the shortest path will skip  
\* encoders and image converters, and it will end in a CSI capture device.  
\* A more complex graph search algorithm could be implemented if a device that  
\* would otherwise be compatible with the pipeline handler isn't correctly  
\* handled by this heuristic.

src/libcamera/pipeline/simple/simple.cpp

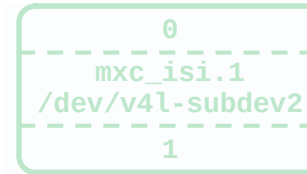
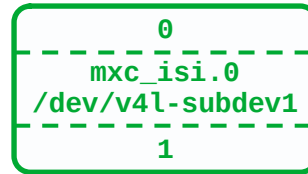
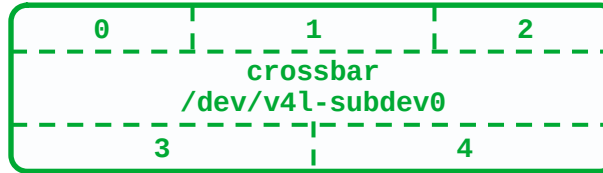
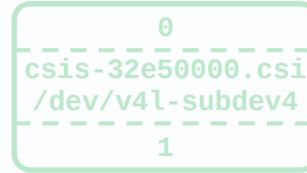
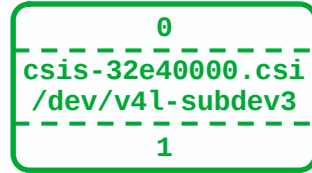
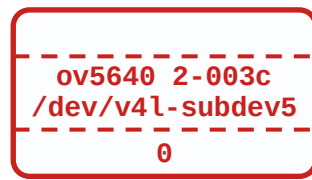


# The Simple Pipeline Handler



# ISI Media Controller Pipeline





# ISI Media Controller Pipeline



```
struct SimplePipelineInfo {  
    const char *driver;  
    /*  
     * Each converter in the list contains the name  
     * and the number of streams it supports.  
     */  
    std::vector<std::pair<const char *, unsigned int>> converters;  
};  
  
static const SimplePipelineInfo supportedDevices[] = {  
    { "imx7-csi", { { "pxp", 1 } } },  
    { "qcom-camss", {} },  
    { "sun6i-csi", {} },  
};
```



# Device Support

```

struct SimplePipelineInfo {
    const char *driver;
    /*
     * Each converter in the list contains the name
     * and the number of streams it supports.
     */
    std::vector<std::pair<const char *, unsigned int>> converters;
};

static const SimplePipelineInfo supportedDevices[] = {
+   { "imx7-csi", { { "pxp", 1 } } },
    { "mxc-isi", {} },
    { "qcom-camss", {} },
    { "sun6i-csi", {} },
};

```



# It's that simple!

```
root@buildroot ~ # cam -l
[0:39:50.151292750] [317] INFO Camera camera_manager.cpp:297 libcamera v0.0.5+59-1f607da9
[0:39:50.336355125] [323] WARN V4L2 v4l2_pixelformat.cpp:285 Unsupported V4L2 pixel
format YM24
Available cameras:
1: Internal front camera (/base/soc@0/bus@30800000/i2c@30a40000/camera@1a)
root@buildroot ~ # cam -c1 -C5
[0:40:46.220308875] [327] INFO Camera camera_manager.cpp:297 libcamera v0.0.5+59-1f607da9
[0:40:46.335423500] [328] WARN V4L2 v4l2_pixelformat.cpp:285 Unsupported V4L2 pixel
format YM24
Using camera /base/soc@0/bus@30800000/i2c@30a40000/camera@3c as cam0
[0:40:46.343990875] [327] INFO Camera camera.cpp:1029 configuring streams: (0) 2592x1944-
YUYV
cam0: Capture 5 frames
979.388197 (0.00 fps) cam0-stream0 seq: 000000 bytesused: 10077696
979.421517 (30.06 fps) cam0-stream0 seq: 000001 bytesused: 10077696
979.454850 (29.99 fps) cam0-stream0 seq: 000002 bytesused: 10077696
979.488182 (30.01 fps) cam0-stream0 seq: 000003 bytesused: 10077696
979.521516 (30.00 fps) cam0-stream0 seq: 000004 bytesused: 10077696
```



# It's that simple!

```
root@buildroot ~ # cam -l
[0:39:50.151292750] [317] INFO Camera camera_manager.cpp:297 libcamera v0.0.5+59-1f607da9
[0:39:50.336355125] [323] WARN V4L2 v4l2_pixelformat.cpp:285 Unsupported V4L2 pixel
format YM24
Available cameras:
1: Internal front camera (/base/soc@0/bus@308000000/i2c@30a40000/camera@1a)
root@buildroot ~ # cam -c1 -C5
[0:40:46.220308875] [327] INFO Camera camera_manager.cpp:297 libcamera v0.0.5+59-1f607da9
[0:40:46.335423500] [328] WARN V4L2 v4l2_pixelformat.cpp:285 Unsupported V4L2 pixel
format YM24
Using camera /base/soc@0/bus@308000000/i2c@30a40000/camera@3c as cam0
[0:40:46.343990875] [327] INFO Camera camera.cpp:1029 configuring streams: (0) 2592x1944-
YUYV
cam0: Capture 5 frames
979.388197 (0.00 fps) cam0-stream0 seq: 000000 bytesused: 10077696
979.421517 (30.06 fps) cam0-stream0 seq: 000001 bytesused: 10077696
979.454850 (29.99 fps) cam0-stream0 seq: 000002 bytesused: 10077696
979.488182 (30.01 fps) cam0-stream0 seq: 000003 bytesused: 10077696
979.521516 (30.00 fps) cam0-stream0 seq: 000004 bytesused: 10077696
```



# Or is it?

```
--- a/src/libcamera/formats.yaml
+++ b/src/libcamera/formats.yaml
@@ -71,6 +71,10 @@ formats:
     fourcc: DRM_FORMAT_YUV422
-   YVU422:
     fourcc: DRM_FORMAT_YVU422
+   YUV444:
+     fourcc: DRM_FORMAT_YUV444
+   YVU444:
+     fourcc: DRM_FORMAT_YVU444

-   MJPEG:
     fourcc: DRM_FORMAT_MJPEG
```



# Adding Missing Formats

```

--- a/src/libcamera/formats.cpp
+++ b/src/libcamera/formats.cpp
@@ -495,6 +495,32 @@ const std::map<PixelFormat, PixelFormatInfo> pixelFormatInfo{
    .pixelsPerGroup = 2,
    .planes = {{ { 2, 1 }, { 1, 1 }, { 1, 1 } }},
} },
+ { formats::YUV444, {
+     .name = "YUV444",
+     .format = formats::YUV444,
+     .v4l2Formats = {
+         .single = V4L2PixelFormat(),
+         .multi = V4L2PixelFormat(V4L2_PIX_FMT_YUV444M),
+     },
+     .bitsPerPixel = 24,
+     .colourEncoding = PixelFormatInfo::ColourEncodingYUV,
+     .packed = false,
+     .pixelsPerGroup = 1,
+     .planes = {{ { 1, 1 }, { 1, 1 }, { 1, 1 } }},
+ } },

/* Greyscale formats. */
{ formats::R8, {

```



# Adding Missing Formats

```

--- a/src/libcamera/v4l2_pixelformat.cpp
+++ b/src/libcamera/v4l2_pixelformat.cpp
@@ -117,6 +117,10 @@ const std::map<V4L2PixelFormat, V4L2PixelFormat::Info> vpf2pf{
    { formats::YUV422, "Planar YUV 4:2:2 (N-C)" } },
    { V4L2PixelFormat(V4L2_PIX_FMT_YVU422M),
      { formats::YVU422, "Planar YVU 4:2:2 (N-C)" } },
+   { V4L2PixelFormat(V4L2_PIX_FMT_YUV444M),
+     { formats::YUV444, "Planar YUV 4:4:4 (N-C)" } },
+   { V4L2PixelFormat(V4L2_PIX_FMT_YUV444M),
+     { formats::YVU444, "Planar YVU 4:4:4 (N-C)" } },

    /* Greyscale formats. */
    { V4L2PixelFormat(V4L2_PIX_FMT_GREY),

```

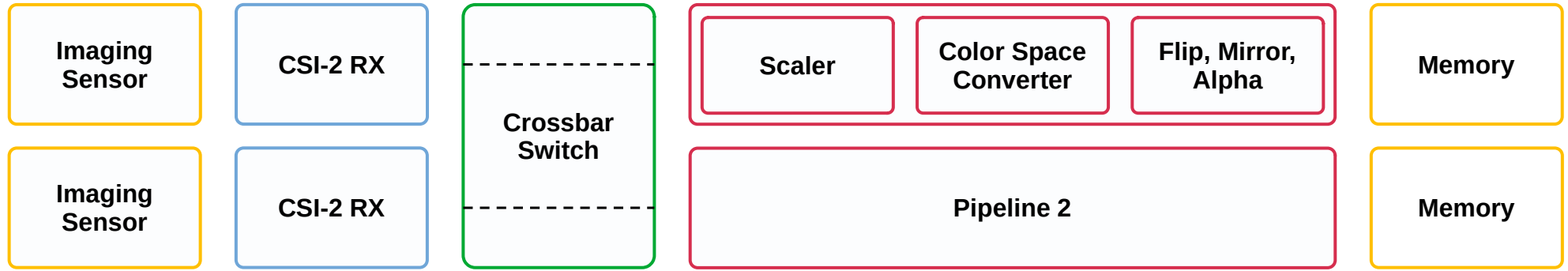


# Adding Missing Formats

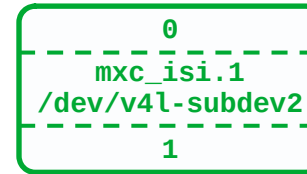
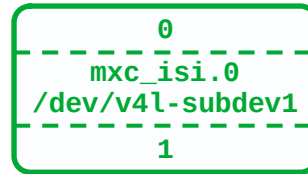
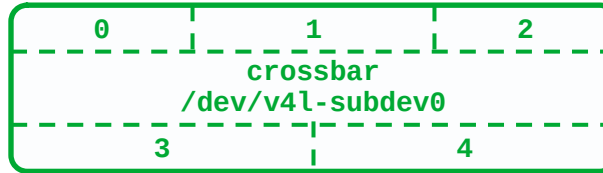
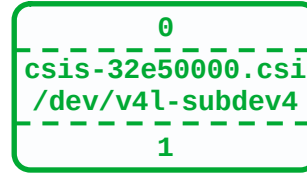
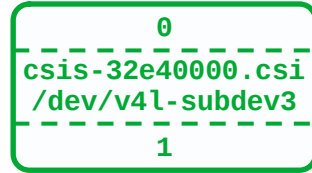
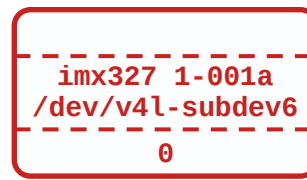
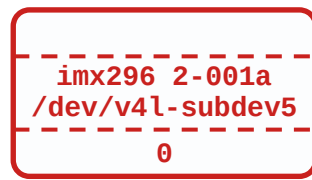
+ - / \ - +  
| (o) |  
+ - - - - +

ISI, Dual  
Camera





## Second Use Case – ISI, Dual Camera



# ISI Media Controller Pipeline

```
root@buildroot ~ # cam -l
[0:33:37.316170875] [262] INFO Camera camera_manager.cpp:297 libcamera v0.0.5+59-1f607da9
Available cameras:
1: Internal front camera (/base/soc@0/bus@30800000/i2c@30a40000/camera@1a)
2: Internal back camera (/base/soc@0/bus@30800000/i2c@30a30000/camera@1a)
root@buildroot ~ # cam -c1 -C5
[0:37:09.068318375] [270] INFO Camera camera_manager.cpp:293 libcamera v0.0.0+3606-
e8ee3e28
Using camera /base/soc@0/bus@30800000/i2c@30a40000/camera@1a as cam0
[0:37:09.216500500] [270] INFO Camera camera.cpp:1029 configuring streams: (0) 1920x1080-
SRGB10
cam0: Capture 5 frames
2229.820427 (0.00 fps) cam0-stream0 seq: 000000 bytesused: 4147200
2229.837072 (60.08 fps) cam0-stream0 seq: 000001 bytesused: 4147200
2229.853748 (59.97 fps) cam0-stream0 seq: 000002 bytesused: 4147200
2229.870414 (60.00 fps) cam0-stream0 seq: 000003 bytesused: 4147200
2229.887077 (60.01 fps) cam0-stream0 seq: 000004 bytesused: 4147200
```

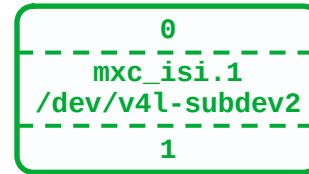
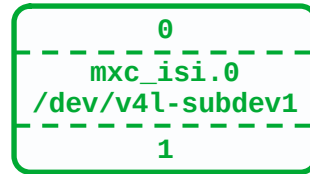
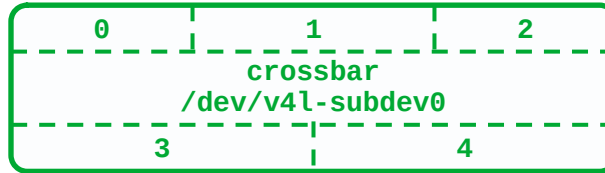
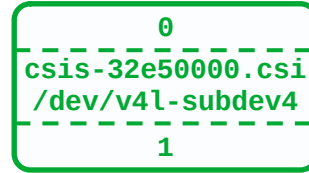
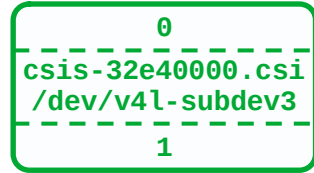
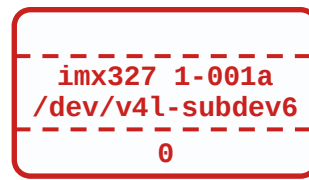
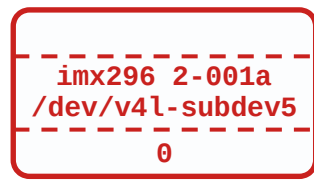


# First camera

```
root@buildroot ~ # cam -c2 -C5
[0:37:09.068318375] [270] INFO Camera camera_manager.cpp:297 libcamera v0.0.5+59-1f607da9
Using camera /base/soc@0/bus@30800000/i2c@30a30000/camera@1a as cam0
[0:41:12.556723875] [275] INFO Camera camera.cpp:1029 configuring streams: (0) 1920x1080-
SRGGB12
[0:41:12.626511375] [276] ERROR V4L2 v4l2_videodevice.cpp:1852 /dev/video0[19:cap]: Failed
to start streaming: Broken pipe
Failed to start capture
Failed to start camera session
```



## Second camera



# ISI Media Controller Pipeline

*We drive MC and V4L2  
standardization and  
extensions development  
according to our needs.*



---

## **Kernel APIs**

*We drive MC and V4L2  
standardization and  
extensions development  
according to our needs.*

*libcamera is a  
userspace framework,  
not a hostile takeover  
of kernel development.*



---

## Kernel APIs

commit 63fce39a74a3156f67909fcda652166183cf6f93

Author: Jacopo Mondì <jacopo@jmondi.org>

Date: Wed Mar 30 12:27:03 2022 +0200

**libcamera: v4l2\_subdevice: Add support for the V4L2 subdev routing API**

Extend the V4L2Subdevice class to support getting and setting routing tables.

Signed-off-by: Jacopo Mondì <jacopo@jmondi.org>

Signed-off-by: Laurent Pinchart <laurent.pinchart@ideasonboard.com>



# V4L2 Subdevice Extension



```

--- a/include/libcamera/internal/v4l2_subdevice.h
+++ b/include/libcamera/internal/v4l2_subdevice.h
@@ -61,6 +61,12 @@ public:
        ActiveFormat = V4L2_SUBDEV_FORMAT_ACTIVE,
};

+   class Routing : public std::vector<struct v4l2_subdev_route>
+   {
+   public:
+       std::string toString() const;
+   };

    explicit V4L2Subdevice(const MediaEntity *entity);
    ~V4L2Subdevice();

@@ -80,6 +86,9 @@ public:
    int setFormat(unsigned int pad, V4L2SubdeviceFormat *format,
                  Whence whence = ActiveFormat);

+   int getRouting(Routing *routing, Whence whence = ActiveFormat);
+   int setRouting(Routing *routing, Whence whence = ActiveFormat);
+

```



# V4L2 Subdevice Extension

commit ba58fbd9ac09587c06de9477c95ebbb1e20d6827

Author: Phi-Bang Nguyen <pnguyen@baylibre.com>

Date: Fri Apr 2 17:00:49 2021 +0200

## **libcamera: pipeline: simple: Walk pipeline using subdev internal routing**

When traversing the media graph to discover a pipeline from the camera sensor to a video node, all sink-to-source paths inside subdevs are considered. This can lead to invalid paths being followed, when a subdev has restrictions on its internal routing.

The V4L2 API supports exposing subdev internal routing to userspace. Make use of this feature, when implemented by a subdev, to restrict the internal paths to the currently active routes. If a subdev doesn't implement the internal routing operations, all source pads are considered, as done today.

This change is needed to properly support multiple sensors with devices such as the NXP i.MX8 ISI or the MediaTek i350 and i500 SENINF. Support for changing routes dynamically will be added later when required.

Signed-off-by: Phi-Bang Nguyen <pnguyen@baylibre.com>

Signed-off-by: Laurent Pinchart <laurent.pinchart@ideasonboard.com>



# **Simple Pipeline Handler Extension**

```

--- a/src/libcamera/pipeline/simple/simple.cpp
+++ b/src/libcamera/pipeline/simple/simple.cpp
@@ -775,6 +798,43 @@ void SimpleCameraData::converterOutputDone(FrameBuffer *buffer)
+/* Retrieve all source pads connected to a sink pad through active routes. */
+std::vector<const MediaPad *> SimpleCameraData::routedSourcePads(MediaPad *sink)
+{
+...
+    V4L2Subdevice::Routing routing = {};
+    ret = subdev->getRouting(&routing, V4L2Subdevice::ActiveFormat);
+    if (ret < 0)
+        return {};
+...
+    for (const struct v4l2_subdev_route &route : routing) {
+        if (sink->index() != route.sink_pad ||
+            !(route.flags & V4L2_SUBDEV_ROUTE_FL_ACTIVE))
+            continue;
+...
+        const MediaPad *pad = entity->getPadByIndex(route.source_pad);
+        pads.push_back(pad);
+    }
+...
+    return pads;
+}

```



# Simple Pipeline Handler Extension

```
$ git log --oneline
```

```
...  
53fd21f1798b libcamera: pipeline: simple: Don't disable links carrying other streams  
ba58fbd9ac09 libcamera: pipeline: simple: Walk pipeline using subdev internal routing  
4756da675dc6 libcamera: pipeline: simple: Setup links in the context of sink entities  
da168e5bf4fc libcamera: pipeline: simple: Reset routing table of subdevs  
63fce39a74a3 libcamera: v4l2_subdevice: Add support for the V4L2 subdev routing API  
06da6ea33b69 libcamera: v4l2_subdevice: Collect subdev capabilities  
2b87018140fd libcamera: v4l2_subdevice: Change V4L2Subdevice::Whence  
ff965168f965 include: linux: Add V4L2 subdev internal routing API  
...
```



# Simple Pipeline Handler Extension

```
root@buildroot ~ # cam -c2 -C5
[0:02:12.088659000] [258] INFO Camera camera_manager.cpp:297 libcamera v0.0.5+59-1f607da9
Using camera /base/soc@0/bus@30800000/i2c@30a30000/camera@1a as cam0
[0:02:12.239059000] [258] INFO Camera camera.cpp:1029 configuring streams: (0) 1456x1088-
SBGGR10
cam0: Capture 5 frames
132.712521 (0.00 fps) cam0-stream0 seq: 000000 bytesused: 3168256
132.729069 (60.43 fps) cam0-stream0 seq: 000001 bytesused: 3168256
132.745629 (60.39 fps) cam0-stream0 seq: 000002 bytesused: 3168256
132.762192 (60.38 fps) cam0-stream0 seq: 000003 bytesused: 3168256
132.778753 (60.38 fps) cam0-stream0 seq: 000004 bytesused: 3168256
```

---

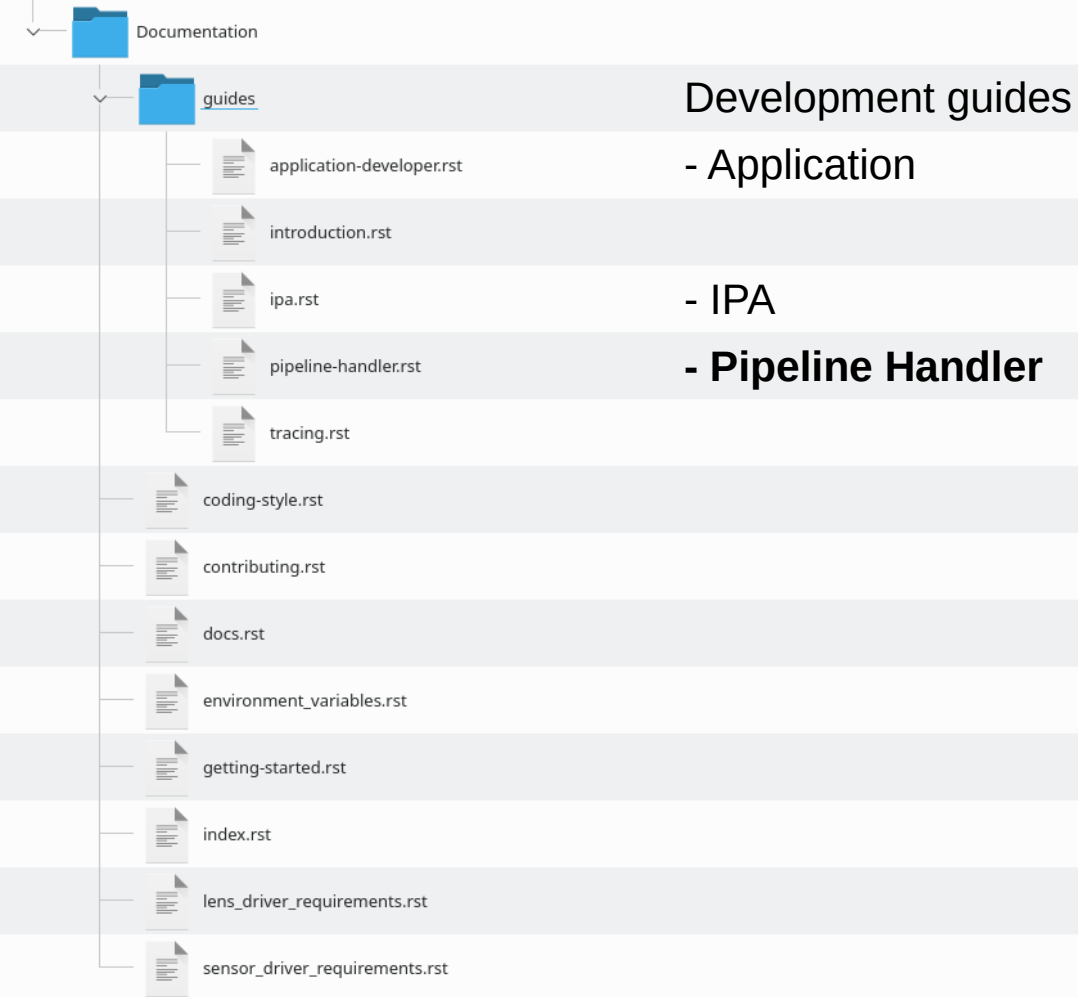
## Second camera



+ - / \ - +  
| ( o ) |  
+ - - - - +

# A Broccoli Pipeline Handler





# Pipeline Handler Writer's Guide

# Pipeline Handler Writers Guide

Pipeline handlers are the abstraction layer for device-specific hardware configuration. They access and control hardware through the V4L2 and Media Controller kernel interfaces, and implement an internal API to control the ISP and capture components of a pipeline directly.

## Prerequisite knowledge: system architecture

A pipeline handler configures and manages the image acquisition and transformation pipeline realized by specialized system peripherals combined with an image source connected to the system through a data and control bus. The presence, number and characteristics of them vary depending on the system design and the product integration of the target platform.

System components can be classified in three macro-categories:

- **Input ports:** Interfaces to external devices, usually image sensors, which transfer data from the physical bus to locations accessible by other system peripherals. An input port needs to be configured according to the input image format and size and could optionally apply basic transformations on the received images, most typically cropping/scaling and some formats conversion. The industry standard for the system typically targeted by libcamera is to have receivers compliant with the MIPI CSI-2 specifications, implemented on a compatible physical layer such as MIPI D-PHY or MIPI C-PHY. Other design are possible but less common, such as LVDS or the legacy BT.601 and BT.656



# Pipeline Handler Writer's Guide



1. The Skeleton
2. The Matching
3. The Configuration
4. The Buffers
5. The Capture



## **Pipeline Handler in 5 Easy Steps**

# 1. The Skeleton

## 2. The Matching

## 3. The Configuration

## 4. The Buffers

## 5. The Capture



- Wire up the build system
- Create the pipeline handler



# The Skeleton

```
--- a/meson_options.txt
+++ b/meson_options.txt
@@ -41,6 +41,7 @@ option('pipelines',
    choices : [
        'all',
        'auto',
+       'brcli',
        'imx8-isi',
        'ipu3',
        'rkisp1',
```

meson\_options.txt



# The Skeleton – Wire Up The Build System

```
# SPDX-License-Identifier: CC0-1.0
```

```
libcamera_sources += files([  
    'brcli.cpp',  
])
```

src/libcamera/pipeline/brcli/meson.build



# The Skeleton – Wire Up The Build System

```

#include "libcamera/internal/pipeline_handler.h"

namespace libcamera {

class PipelineHandlerBrcli : public PipelineHandler
{
Public:
    PipelineHandlerBrcli(CameraManager *manager);

    bool match(DeviceEnumerator *enumerator) override;
    CameraConfiguration *generateConfiguration(Camera *camera,
                                                const StreamRoles &roles) override;
    int configure(Camera *camera, CameraConfiguration *config) override;
    int exportFrameBuffers(Camera *camera, Stream *stream,
                           std::vector<std::unique_ptr<FrameBuffer>> *buffers) override;
    int start(Camera *camera, const ControlList *controls) override;

protected:
    int queueRequestDevice(Camera *camera, Request *request) override;
    void stopDevice(Camera *camera) override;
};

REGISTER_PIPELINE_HANDLER(PipelineHandlerBrcli)

} /* namespace libcamera */

```

src/libcamera/pipeline/brcli/brcli.cpp



# The Skeleton – Create The Pipeline Handler

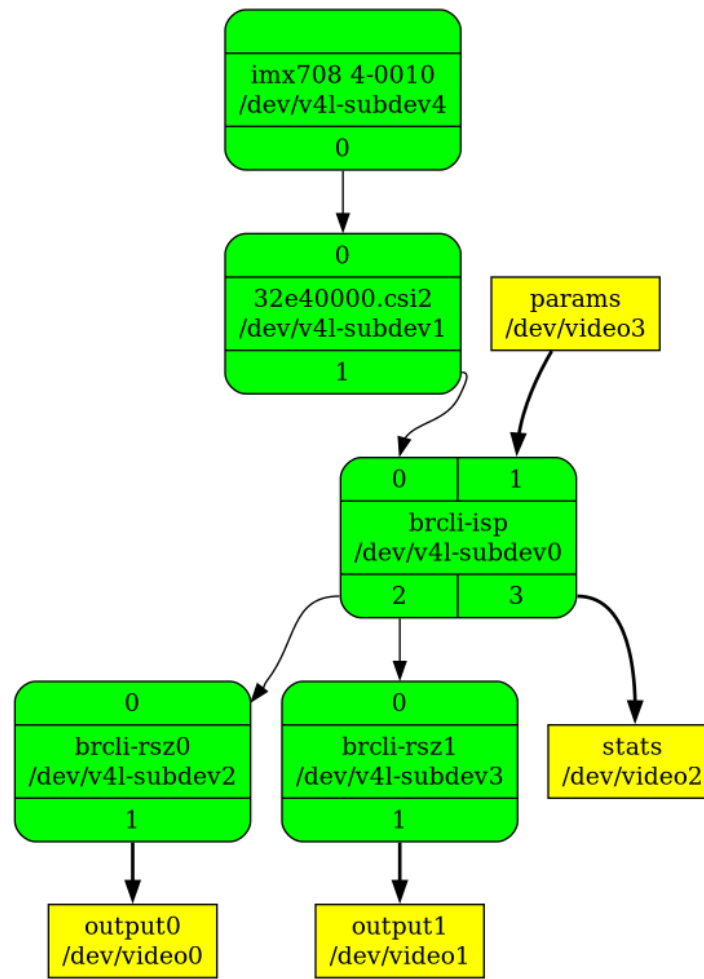
1. The Skeleton
- 2. The Matching**
3. The Configuration
4. The Buffers
5. The Capture



- Match media devices
- Discover the pipeline
- Create the camera data
- Register the cameras



## The Matching



# Media Graph

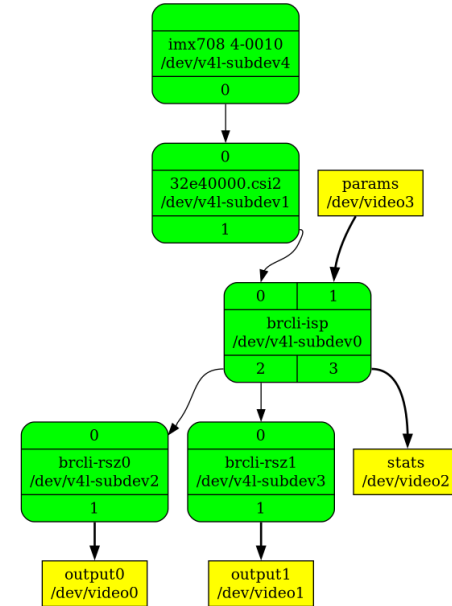


```
PipelineHandlerBrcli::PipelineHandlerBrcli(CameraManager *manager)
    : PipelineHandler(manager)
{
}
```

```
bool PipelineHandlerBrcli::match(DeviceEnumerator *enumerator)
{
    DeviceMatch dm("brcli-cam");
    dm.add("brcli-isp");
    dm.add("brcli-rsz0");
    dm.add("brcli-rsz1");

    mediaDev_ = acquireMediaDevice(enumerator, dm);
    if (!mediaDev_)
        return false;

    return false;
}
```



src/libcamera/pipeline/brcli/brcli.cpp



# The Matching – Match Media Devices



```

#include <memory>
#include <vector>

#include "libcamera/internal/media_device.h"
#include "libcamera/internal/v4l2_subdevice.h"
#include "libcamera/internal/v4l2_videodevice.h"

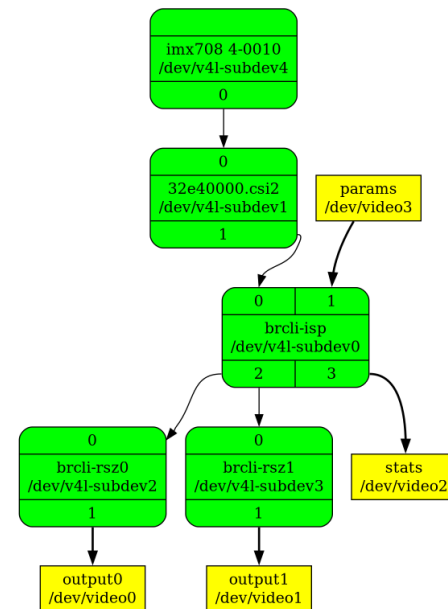
...

class PipelineHandlerBrcli : public PipelineHandler
{
...
private:
    struct Path {
        std::unique_ptr<V4L2Subdevice> resizer;
        std::unique_ptr<V4L2VideoDevice> video;
        bool isEnabled;
    };

    MediaDevice *mediaDev_;

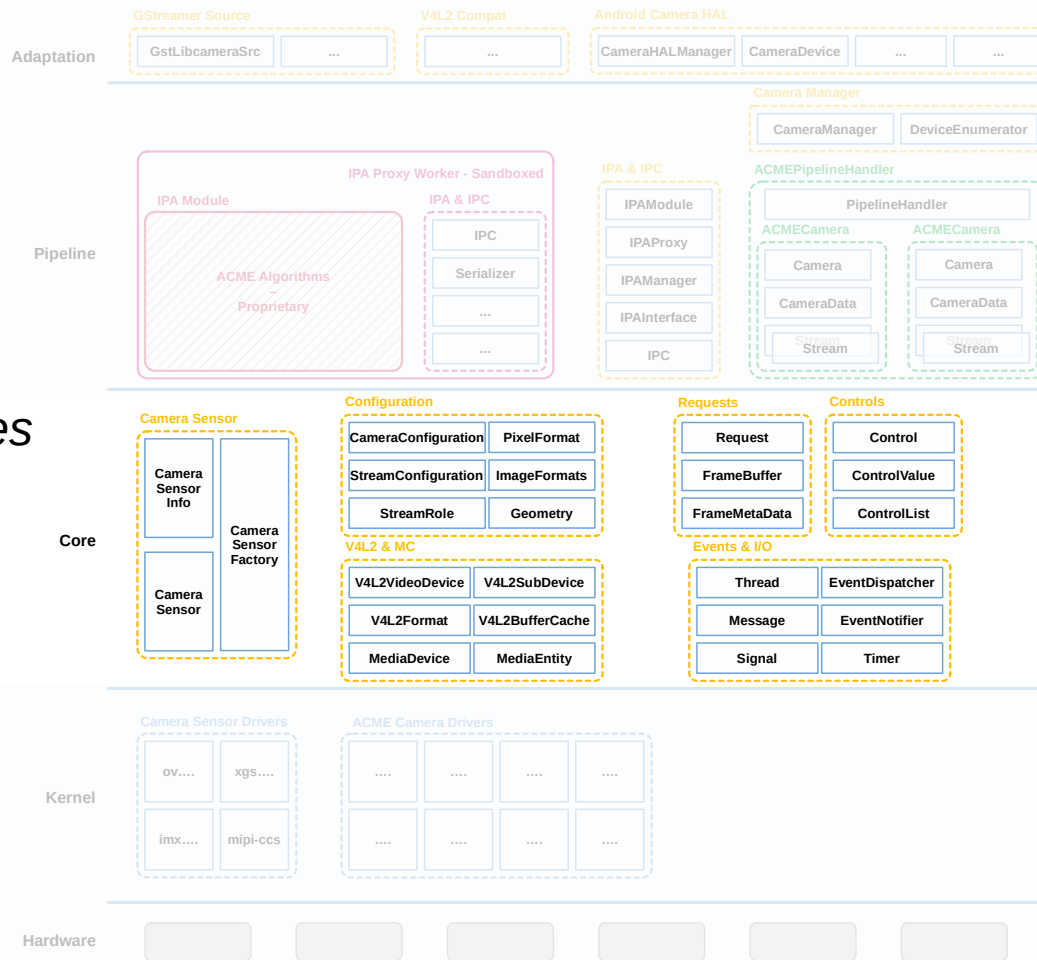
    std::unique_ptr<V4L2Subdevice> sensor_;
    std::unique_ptr<V4L2Subdevice> csi2rx_;
    std::unique_ptr<V4L2Subdevice> isp_;
    std::array<Path, 2> paths_;
};

```



src/libcamera/pipeline/brcli/brcli.cpp

# The Matching – Discover The Pipeline



*Many helper classes ease the implementation of pipeline handlers.*

# The Helpers and Support Classes



```

bool PipelineHandlerBrcli::match(DeviceEnumerator *enumerator)
{
    ...

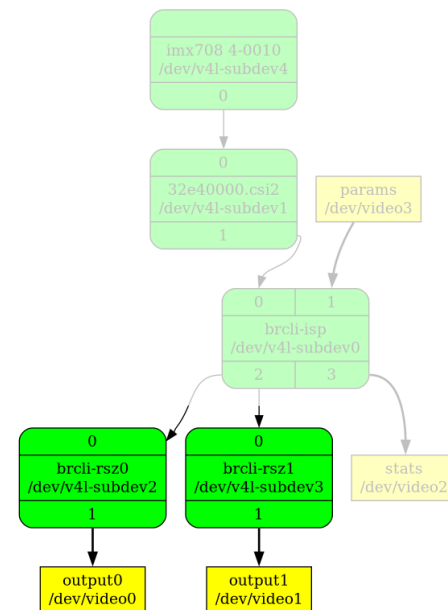
    /* Acquire the subdevs and video nodes for the resizers. */
    for (unsigned int i = 0; i < 2; ++i) {
        std::string entityName = "brcli-rsz." + std::to_string(i);
        std::unique_ptr<V4L2Subdevice> rsz =
            V4L2Subdevice::fromEntityName(mediaDev_, entityName);
        if (!rsz)
            return false;

        entityName = "output" + std::to_string(i);
        std::unique_ptr<V4L2VideoDevice> video =
            V4L2VideoDevice::fromEntityName(mediaDev_, entityName);
        if (!video)
            return false;

        paths_[i] = { std::move(rsz), std::move(video) };
    }

    return false;
}

```



src/libcamera/pipeline/brcli/brcli.cpp

# The Matching – Discover The Pipeline



```

bool PipelineHandlerBrcli::match(DeviceEnumerator *enumerator)
{
    ...

    /* Acquire the subdev for the ISP. */
    isp_ = V4L2Subdevice::fromEntityName(mediaDev_, "brcli-isp");
    if (!isp_)
        return false;

    if (isp_>entity()->pads().size() != 4)
        return false;

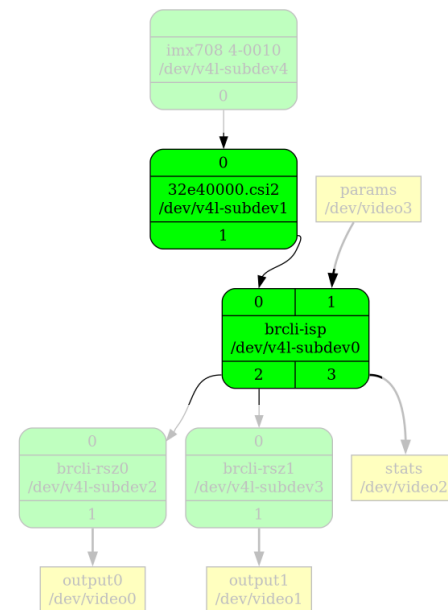
    /* Find the CSI-2 receiver connected to the ISP input. */
    MediaPad *pad = isp_>entity()->pads()[0];

    MediaEntity *csi = pad->links()[0]->source()->entity();
    if (csi->pads().size() != 2)
        return false;

    csi2rx_ = std::make_unique<V4L2Subdevice>(csi);

    return false;
}

```



src/libcamera/pipeline/brcli/brcli.cpp



# The Matching – Discover The Pipeline

```

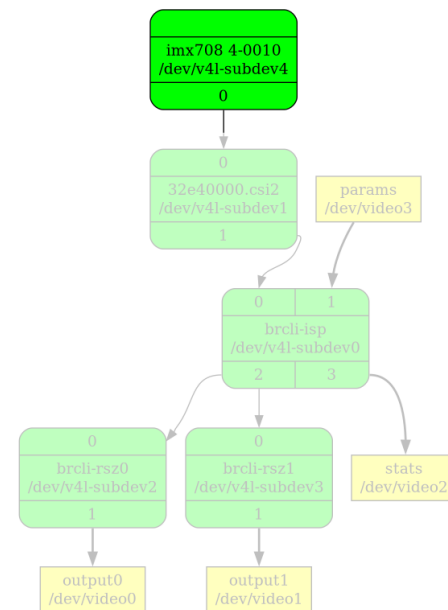
bool PipelineHandlerBrcli::match(DeviceEnumerator *enumerator)
{
    ...

    /* Find the camera sensor connected to the CSI-2 receiver. */
    pad = csi->pads()[0];
    if (!(pad->flags() & MEDIA_PAD_FL_SINK) || pad->links().empty())
        return false;

    MediaEntity *sensor = pad->links()[0]->source()->entity();
    if (sensor->function() != MEDIA_ENT_F_CAM_SENSOR)
        return false;

    return false;
}

```



src/libcamera/pipeline/brcli/brcli.cpp



# The Matching – Discover The Pipeline

```
class BrcliCameraData : public Camera::Private
{
public:
    BrcliCameraData(PipelineHandler *ph)
        : Camera::Private(ph)
    {
    }

    PipelineHandlerBrcli *pipe()
    {
        return static_cast<PipelineHandlerBrcli *>(Camera::Private::pipe());
    }

    CameraSensor *sensor_;

    std::array<Stream, 2> streams_;
};
```

src/libcamera/pipeline/brcli/brcli.cpp



## The Matching – Create The Camera Data

```

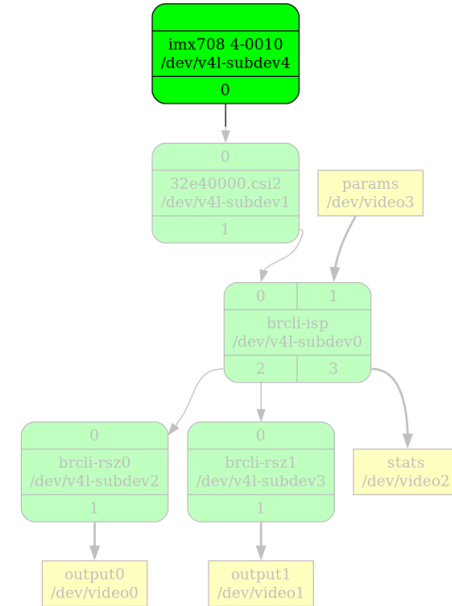
bool PipelineHandlerBrcli::match(DeviceEnumerator *enumerator)
{
    ...

    /* Create the camera data. */
    std::unique_ptr<BrcliCameraData> data =
        std::make_unique<BrcliCameraData>(this);

    data->sensor_ = std::make_unique<CameraSensor>(sensor);

    return false;
}

```



src/libcamera/pipeline/brcli/brcli.cpp

# The Matching – Create The Camera Data



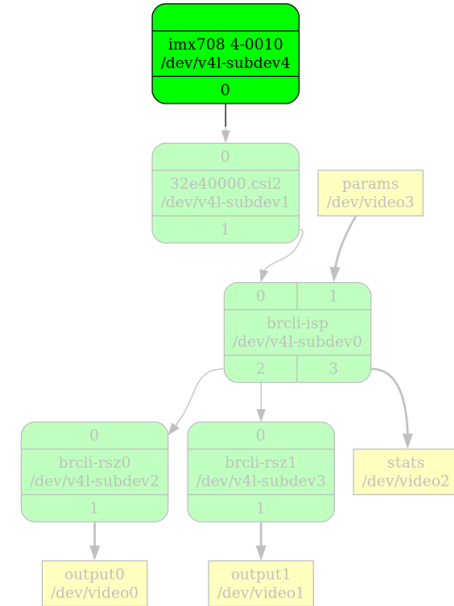
```

class BrcliCameraData : public Camera::Private
{
public:
    ...
    int init();
};

int BrcliCameraData::init()
{
    int ret = sensor_->init();
    if (ret)
        return ret;

    return 0;
}

```



src/libcamera/pipeline/brcli/brcli.cpp



# The Matching – Create The Camera Data



```

bool PipelineHandlerBrcli::match(DeviceEnumerator *enumerator)
{
    ...

    ret = csi2rx_>open();
    if (ret)
        return false;

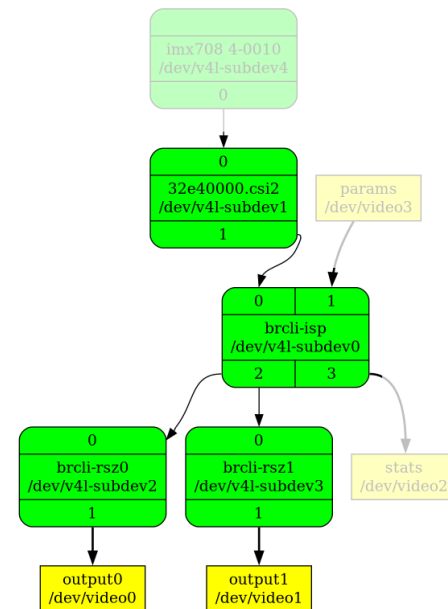
    ret = isp_>open();
    if (ret)
        return false;

    for (auto path : paths_) {
        ret = path.resizer->open();
        if (ret)
            return false;

        ret = path.video->open();
        if (ret)
            return false;
    }

    return false;
}

```



src/libcamera/pipeline/brcli/brcli.cpp



# The Matching – Complete Initialization

```

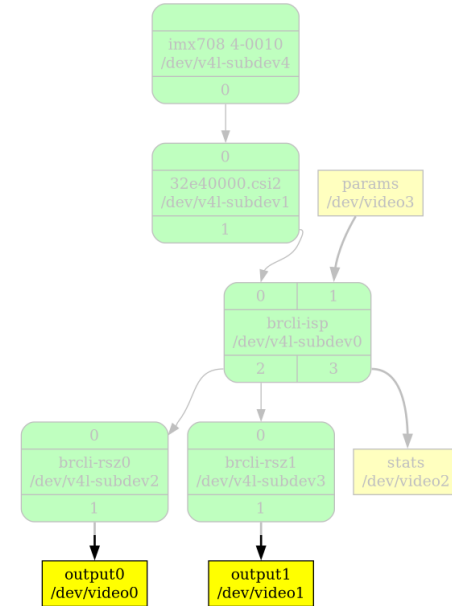
bool PipelineHandlerBrcli::match(DeviceEnumerator *enumerator)
{
    ...

    /* Register the camera. */
    const std::string &id = data->sensor_->id();
    std::set<Stream *> streams = { &data->streams_[0], &data->streams_[1] };
    std::shared_ptr<Camera> camera =
        Camera::create(std::move(data), id, streams);

    registerCamera(std::move(camera));

    return true;
}

```



src/libcamera/pipeline/brcli/brcli.cpp



# The Matching – Register The Camera

```
root@buildroot ~ # cam -l  
[0:01:28.355935500] [257] INFO Camera camera_manager.cpp:297 libcamera v0.0.5+59-1f607da9  
Available cameras:  
1: (/base/soc@0/bus@30800000/i2c@30a40000/camera@1a)
```



## Intermezzo: First Test

```
int BrcliCameraData::init()  
{  
    ...  
    properties_ = pipe_.sensor->properties();  
    return 0;  
}
```

src/libcamera/pipeline/brcli/brcli.cpp



## Intermezzo: Add Camera Properties

```
root@buildroot ~ # cam -l  
[0:03:42.037213125] [260] INFO Camera camera_manager.cpp:297 libcamera v0.0.5+59-1f607da9  
Available cameras:  
1: Internal front camera (/base/soc@0/bus@30800000/i2c@30a40000/camera@1a)
```



## Intermezzo: Second Test

1. The Skeleton
2. The Matching
- 3. The Configuration**
4. The Buffers
5. The Capture



- Generate a configuration
- Validate the configuration
- Configure the camera



## The Configuration

```

CameraConfiguration *
PipelineHandlerBrcli::generateConfiguration(Camera *camera,
                                           const StreamRoles &roles)
{
    BrcliCameraData *data = cameraData(camera);
    CameraConfiguration *config = new BrcliCameraConfiguration(data);

    if (roles.empty())
        return config;

    /* \todo Customize the configuration based on hardware capabilities. */
    StreamConfiguration cfg;
    cfg.size = { 1920, 1080 };
    cfg.pixelFormat = formats::YUYV;
    cfg.bufferCount = 4;

    config->addConfiguration(cfg);
    config->validate();

    return config;
}

```

src/libcamera/pipeline/brcli/brcli.cpp



# The Configuration – Generate

```
class BrcliCameraConfiguration : public CameraConfiguration
{
public:
    BrcliCameraConfiguration(BrcliCameraData *data)
        : data_(data)
    {
    }

    Status validate() override;

private:
    const BrcliCameraData *data_;
};
```

src/libcamera/pipeline/brcli/brcli.cpp



# The Configuration – Generate



```

CameraConfiguration::Status BrcliCameraConfiguration::validate()
{
    Status status = Valid;

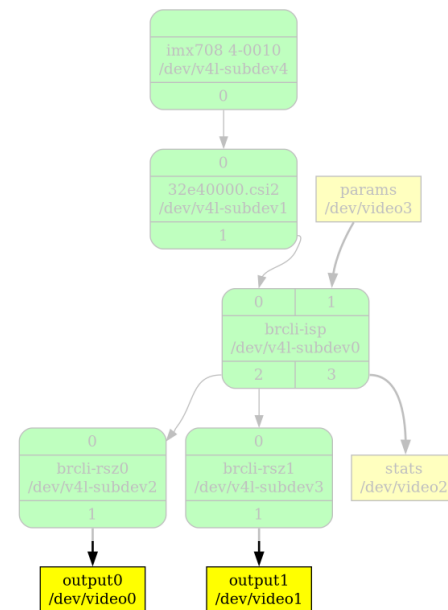
    if (config_.empty())
        return Invalid;

    if (config_.size() > 2) {
        config_.resize(2);
        status = Adjusted;
    }

    for (StreamConfiguration &cfg : config_) {
        /* \todo Validate the configuration. */
    }

    return status;
}

```



src/libcamera/pipeline/brcli/brcli.cpp



# The Configuration – Validate

```

int PipelineHandlerBrcli::configure(Camera *camera, CameraConfiguration *c)
{
    BrcliCameraConfiguration *camConfig = static_cast<BrcliCameraConfiguration *>(c);

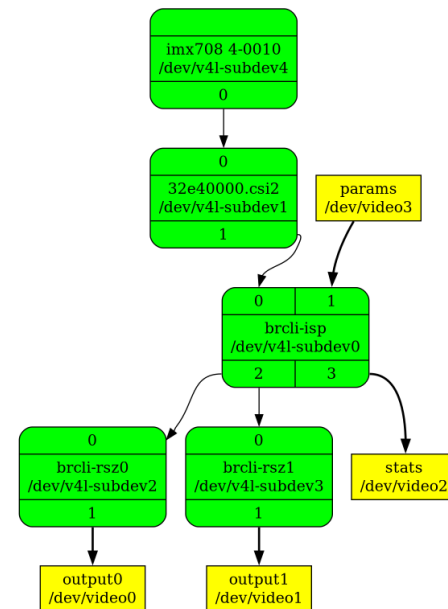
    BrcliCameraData *data = cameraData(camera);
    CameraSensor *sensor = data->sensor_.get();

    /* All links are immutable except the sensor -> csis link. */
    const MediaPad *sensorSrc = sensor->entity()->getPadByIndex(0);
    sensorSrc->links()[0]->setEnabled(true);

    ...

    return 0;
}

```



src/libcamera/pipeline/brcli/brcli.cpp



# The Configuration – Configure

```

int PipelineHandlerBrcli::configure(Camera *camera, CameraConfiguration *c)
{
...
    /* Apply formats to the capture pipeline. */
    StreamConfiguration &config = camConfig->at(0);

...

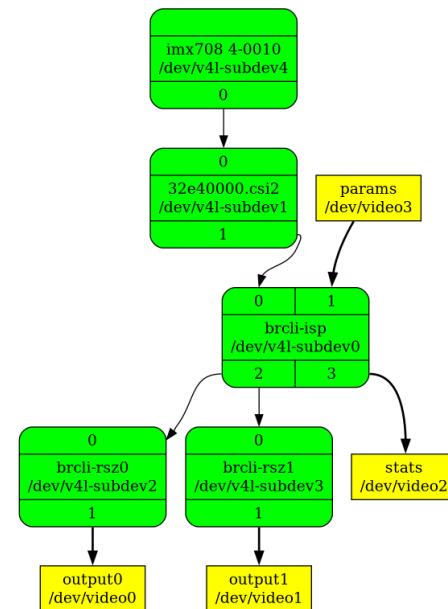
    V4L2SubdeviceFormat sensorFmt{};
    sensorFmt.mbus_code = pipeFormat.mbusCode;
    sensorFmt.size = config.size;
    int ret = sensor->setFormat(&sensorFmt);
    if (ret)
        return ret;

    ret = csi2rx_->setFormat(0, &sensorFmt);
    if (ret)
        return ret;

    ret = csi2rx_->setFormat(1, &sensorFmt);
    if (ret)
        return ret;

...
    return 0;
}

```



src/libcamera/pipeline/brcli/brcli.cpp

# The Configuration – Configure

1. The Skeleton
2. The Matching
3. The Configuration
- 4. The Buffers**
5. The Capture



- Export frame buffers



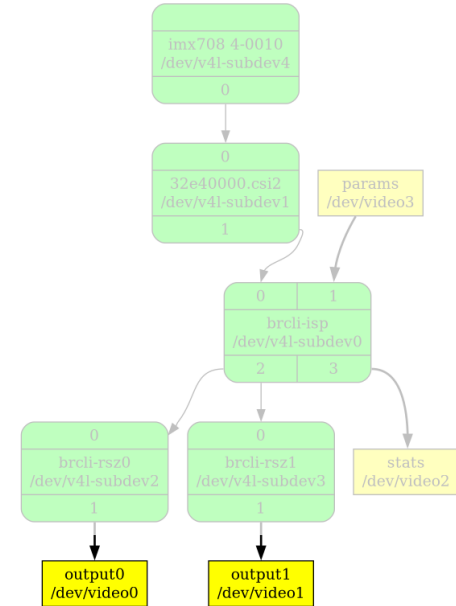
## The Buffers

```

int PipelineHandlerBrcli::exportFrameBuffers(Camera *camera, Stream *stream,
                                             std::vector<std::unique_ptr<FrameBuffer>> *buffers)
{
    BrcliCameraData *data = cameraData(camera);
    unsigned int count = stream->configuration().bufferCount;

    return paths_[...].video->exportBuffers(count, buffers);
}

```



src/libcamera/pipeline/brcli/brcli.cpp



# The Buffers – Export Frame Buffers

1. The Skeleton
2. The Matching
3. The Configuration
4. The Buffers
5. The Capture



- Start the device
- Queue capture requests
- Stop the device



## The Capture

```

int PipelineHandlerBrcli::start(Camera *camera,
                                [[maybe_unused]] const ControlList *controls)
{
    BrcliCameraData *data = cameraData(camera);

    for (auto path : paths_) {
        if (!path.isEnabled)
            continue;

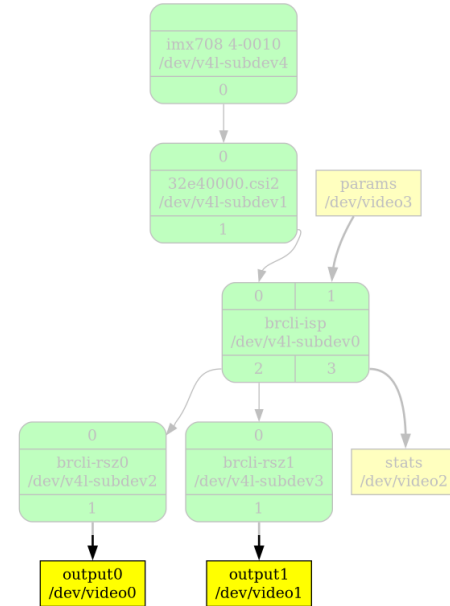
        V4L2VideoDevice *video = path.video.get();
        const Stream *stream = &streams_[...];
        const StreamConfiguration &config = stream->configuration();

        int ret = video->importBuffers(config.bufferCount);
        if (ret)
            return ret;

        ret = video->streamOn();
        if (ret)
            return ret;
    }

    return 0;
}

```



# The Capture – Start The Device



```

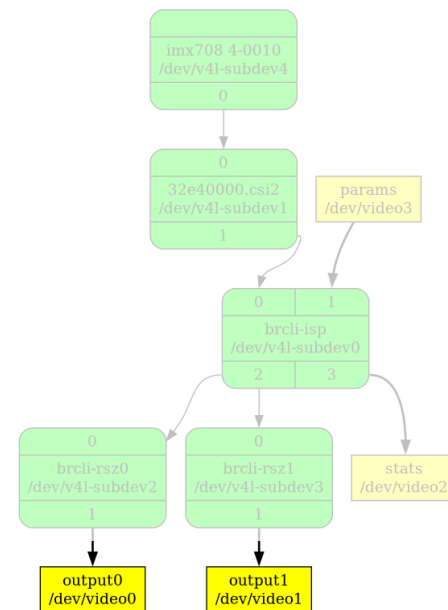
int PipelineHandlerBrcli::queueRequestDevice(Camera *camera, Request *request)
{
    BrcliCameraData *data = cameraData(camera);

    for (auto &[stream, buffer] : request->buffers()) {
        V4L2VideoDevice *capture = paths_[...].video.get();

        int ret = capture->queueBuffer(buffer);
        if (ret)
            return ret;
    }

    return 0;
}

```



src/libcamera/pipeline/brcli/brcli.cpp



# The Capture – Queue Capture Requests

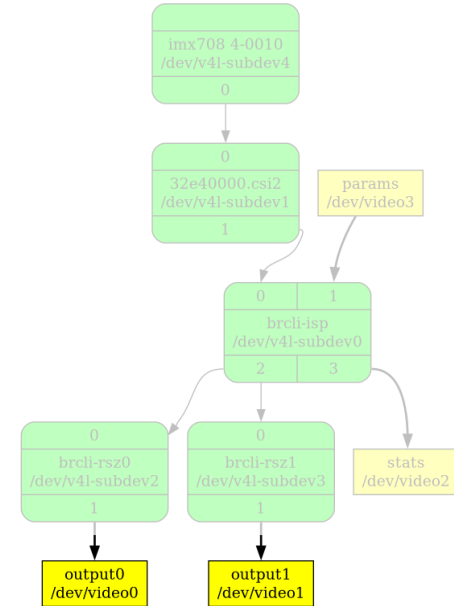


```

void PipelineHandlerBrcli::stopDevice(Camera *camera)
{
    for (auto path : paths_) {
        if (!path.isEnabled)
            continue;

        V4L2VideoDevice *video = path.video.get();
        video->streamOff();
        video->releaseBuffers();
    }
}

```



src/libcamera/pipeline/brcli/brcli.cpp

# The Capture – Stop The Device



```
root@buildroot ~ # cam -c1 -C5
[1:33:46.502527375] [298] INFO Camera camera_manager.cpp:297 libcamera v0.0.5+59-1f607da9
Using camera /base/soc@0/bus@30800000/i2c@30a40000/camera@1a as cam0
[1:33:46.674367625] [298] INFO Camera camera.cpp:1029 configuring streams: (0) 1920x1080-
SRGGB10
cam0: Capture 5 frames
```



# Final Test

```
root@buildroot ~ # cam -c1 -C5
[1:33:46.502527375] [298] INFO Camera camera_manager.cpp:297 libcamera v0.0.5+59-1f607da9
Using camera /base/soc@0/bus@30800000/i2c@30a40000/camera@1a as cam0
[1:33:46.674367625] [298] INFO Camera camera.cpp:1029 configuring streams: (0) 1920x1080-
SRGGB10
cam0: Capture 5 frames
<wait>
```



# Final Test

```
root@buildroot ~ # cam -c1 -C5
[1:33:46.502527375] [298] INFO Camera camera_manager.cpp:297 libcamera v0.0.5+59-1f607da9
Using camera /base/soc@0/bus@30800000/i2c@30a40000/camera@1a as cam0
[1:33:46.674367625] [298] INFO Camera camera.cpp:1029 configuring streams: (0) 1920x1080-
SRGGB10
cam0: Capture 5 frames
<wait>
<wait some more>
```



# Final Test

```
root@buildroot ~ # cam -c1 -C5
[1:33:46.502527375] [298] INFO Camera camera_manager.cpp:297 libcamera v0.0.5+59-1f607da9
Using camera /base/soc@0/bus@30800000/i2c@30a40000/camera@1a as cam0
[1:33:46.674367625] [298] INFO Camera camera.cpp:1029 configuring streams: (0) 1920x1080-
SRGGB10
cam0: Capture 5 frames
<wait>
<wait some more>
<get a cup of tea>
```



# Final Test

```

bool PipelineHandlerBrcli::match(DeviceEnumerator *enumerator)
{
    ...
    for (unsigned int i = 0; ; ++i) {
        ...
        std::unique_ptr<V4L2VideoDevice> video =
            V4L2VideoDevice::fromEntityName(mediaDev_, entityName);
        if (!video)
            Break;

        video->bufferReady.connect(this, &PipelineHandlerBrcli::bufferReady);

        paths_[i] = { std::move(rsz), std::move(video) };
    }

    ...
}

void PipelineHandlerBrcli::bufferReady(FrameBuffer *buffer)
{
    Request *request = buffer->request();

    completeBuffer(request, buffer);
    completeRequest(request);
}

```

src/libcamera/pipeline/brcli/brcli.cpp



# The Capture – Handle Events

```
root@buildroot ~ # cam -c1 -C5
```

```
[1:35:17.092654500] [300] INFO Camera camera_manager.cpp:297 libcamera v0.0.5+59-1f607da9
```

```
Using camera /base/soc@0/bus@30800000/i2c@30a40000/camera@1a as cam0
```

```
[1:35:17.263601750] [300] INFO Camera camera.cpp:1029 configuring streams: (0) 1920x1080-SRGGB10
```

**cam0: Capture 5 frames**

```
5717.881212 (0.00 fps) cam0-stream0 seq: 000000 bytesused: 4147200
```

```
5717.897881 (59.99 fps) cam0-stream0 seq: 000001 bytesused: 4147200
```

```
5717.914544 (60.01 fps) cam0-stream0 seq: 000002 bytesused: 4147200
```

```
5717.931211 (60.00 fps) cam0-stream0 seq: 000003 bytesused: 4147200
```

```
5717.947879 (60.00 fps) cam0-stream0 seq: 000004 bytesused: 4147200
```



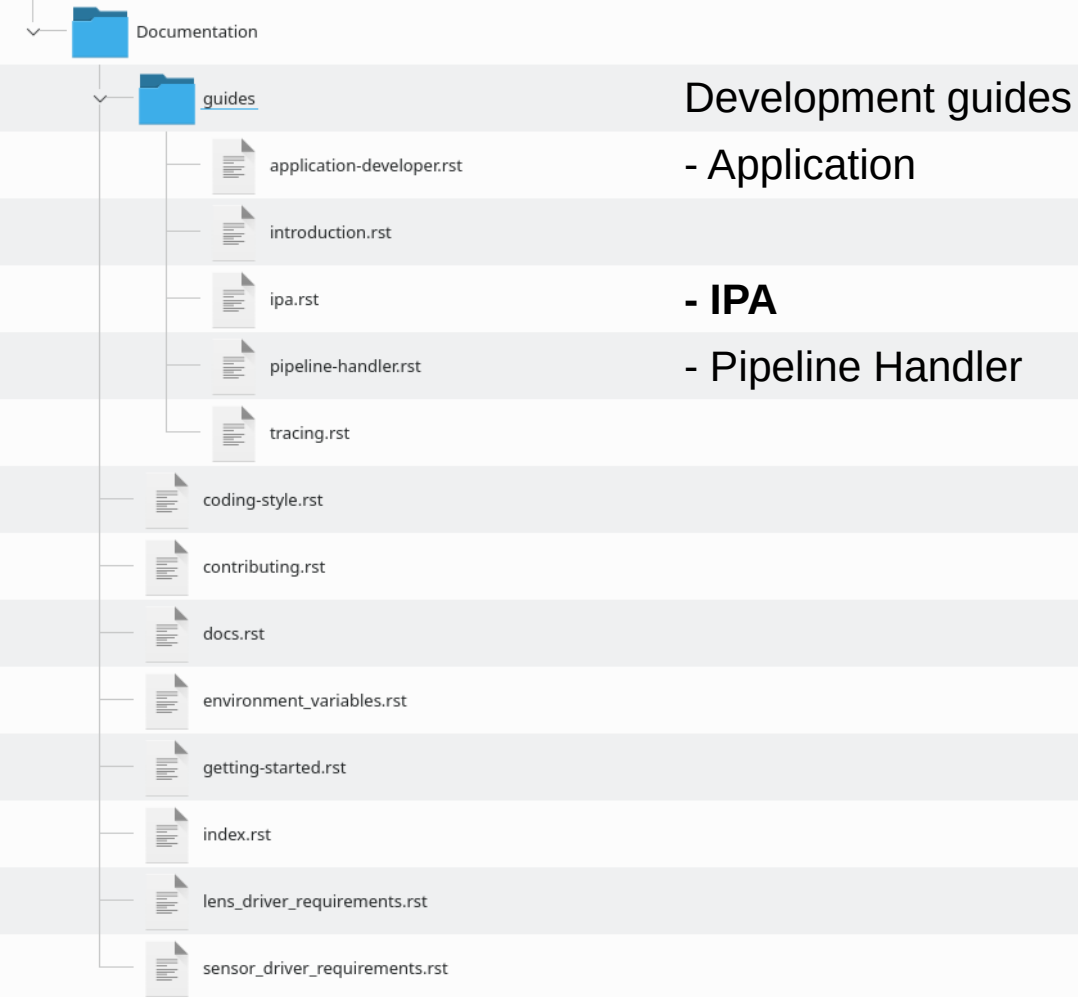
# Final Test, For Real

+ - / \ - +  
| (o) |  
+ - - - - +

# Controlling The ISP







# IPA Writer's Guide

# IPA Writer's Guide

IPA modules are Image Processing Algorithm modules. They provide functionality that the pipeline handler can use for image processing.

This guide covers the definition of the IPA interface, and how to plumb the connection between the pipeline handler and the IPA.

## The IPA interface and protocol

The IPA interface defines the interface between the pipeline handler and the IPA. Specifically, it defines the functions that the IPA exposes that the pipeline handler can call, and the signals that the pipeline handler can connect to, in order to receive data from the IPA asynchronously. In addition, it contains any custom data structures that the pipeline handler and IPA may pass to each other.

It is possible to use the same IPA interface with multiple pipeline handlers on different hardware platforms. Generally in such cases, these platforms would have a common hardware



# IPA Writer's Guide

1. The IPA Interface
2. The Pipeline Handler
3. The IPA Module
4. The Algorithms



## **IPA Module in 4 Easy Steps**

# 1. The IPA Interface

## 2. The Pipeline Handler

## 3. The IPA Module

## 4. The Algorithms



- Define the interface between the Pipeline Handler and IPA Module



# The IPA Interface

```

module ipa.brcli;

import "include/libcamera/ipa/core.mojom";

interface IPABrcliInterface {
    init(libcamera.IPASettings settings,
        uint32 hwRevision,
        libcamera.IPACameraSensorInfo sensorInfo,
        libcamera.ControlInfoMap sensorControls)
        => (int32 ret, libcamera.ControlInfoMap ipaControls);
    start() => (int32 ret);
    stop();

    configure(IPAConfigInfo configInfo,
        map<uint32, libcamera.IPASStream> streamConfig)
        => (int32 ret, libcamera.ControlInfoMap ipaControls);

    mapBuffers(array<libcamera.IPABuffer> buffers);
    unmapBuffers(array<uint32> ids);

    [async] queueRequest(uint32 frame, libcamera.ControlList reqControls);
    [async] fillParamsBuffer(uint32 frame, uint32 bufferId);
    [async] processStatsBuffer(uint32 frame, uint32 bufferId,
        libcamera.ControlList sensorControls);
};

```

include/libcamera/ipa/brcli.mojom



# The IPA Interface

```
module ipa.brcli;  
  
import "include/libcamera/ipa/core.mojom";  
  
...  
  
interface IPABrcliEventInterface {  
    paramsBufferReady(uint32 frame);  
    setSensorControls(uint32 frame, libcamera.ControlList sensorControls);  
    metadataReady(uint32 frame, libcamera.ControlList metadata);  
};
```

include/libcamera/ipa/brcli.mojom



# The IPA Interface

1. The IPA Interface

2. The Pipeline Handler ▶

3. The IPA Module

4. The Algorithms

- Wire up the statistics capture and ISP parameters
- Communicate with the IPA module



## The Pipeline Handler

See you at the next  
conference for the rest  
of the story.



---

## The Pipeline Handler



See you at the next  
conference for the rest  
of the story.

If you really can't wait,  
contact me for spoilers.



---

## The Pipeline Handler

+ - / \ - +

| (o) |

+ - - - - +

libcamera



`libcamera-devel@lists.libcamera.org`

`irc://chat.freenode.net/#libcamera`

`laurent.pinchart@ideasonboard.com`



## Contact

?

!



**By the way, we are hiring  
[jobs@ideasonboard.com](mailto:jobs@ideasonboard.com)**



děkuji

